بسمه تعالی

بخش اول

کارایی ، تحلیل و مرتبه الگوریتم ها، نمادهای مجانبی

ارائه دهنده : محمد فرجی مهماندار

تعریف الگوریتم

- به زبان ساده می‌توان گفت الگوریتم مجموعه‌ای از دستورالعمل‌هاست که اگر به ترتیب دنبال شوند، موجب حل مسأله می‌گردند. ترتیب مراحل و شرط خاتمه عملیات باید کاملا مشخص باشد
- خصوصیات هر الگوریتم:
  - ورودی: می تواند ورودی داشته باشد یا نداشته باشد
  - خروجی: حداقل باید دارای یک خروجی باشد.
  - دارای ترتیب (توالی) است.
  - واضح و صریح است.(بدون ابهام می باشد).
  - محدود است.
- **طراحی الگوریتم :** منظور از طراحی الگوریتم کشف الگوریتم، ایجاد، تعیین اعتبار، آنالیز و ارزیابی الگوریتم برای یک مسأله می‌باشد.
- **آنالیز الگوریتم :** منظور از آنالیز الگوریتم این است که پارامترهای زمان محاسبه و حافظه مورد نیاز برای محاسبه الگوریتم به دست آید تا بتوان، الگوریتم‌های مختلف را برای یک مسأله خاص با یکدیگر مقایسه کرد

هدف از مطالعه درس طراحی الگوریتم

- به طور معمول برای حل یک مسئله مشخص بیش از یک الگوریتم وجود دارد.
- برخی از این الگوریتم ها از بقیه کاراتر می باشند.
- الگوریتمی انتخاب می شود که بیشترین کارایی را داشته باشد.
- چه مواردی را باید بررسی کنیم:
  - آیا الگوریتمی که ارائه می شود درست است یا خیر؟
  - چگونه الگوریتم های مربوط به یک مسئله مقایسه می شوند؟
  - کارایی (efficiency) هر الگوریتم چگونه تعیین می شود؟
    - برای این کار نیاز به تحلیل الگوریتم ها داریم.
- **مطالعه الگوریتم ها**

۱) طراحی الگوریتم

۲) اثبات درستی یا معتبر سازی الگوریتم

۳) بیان یا پیاده سازی الگوریتم

۴) تحلیل الگوریتم

بحث ما در این درس : ۱ و ۴

تعریف مسئله

- مسئله: سؤالی که به دنبال پاسخ آن هستیم.
  - مسئله (A): مرتب سازی یک لیست S متشکل از n عدد به ترتیب غیر نزولی.
  - پاسخ دنباله مرتب از n عدد می باشد.

Sorting

- Input: Sequence of n numbers $\langle a_1, \ldots, a_n \rangle$
- Output: Permutation (reordering)

$$\langle a'_1, a'_2, \ldots, a'_n \rangle$$

such that $a'_1 \le a'_2 \le \ldots \le a'_n$

پارامترهای مسئله

- مسئله(B): تعیین اینکه آیا عدد x در لیست S متشکل از n عدد وجود دارد یا خیر. در صورت وجود پاسخ "بله" و در غیر این صورت پاسخ برابر با "خیر" خواهد بود.

نمونه مسئله

- مسائلی که شامل پارامترها هستند بیانگر کلاسی از مسائل می باشند.
  - نمونه مسئله: هر انتساب خاصی از مقادیر به پارامترها
  - راه حل یک نمونه مسأله: پاسخ سؤال پرسیده شده توسط مسأله در آن نمونه مسئله
- مساله A
  - نمونه: $S = [10, 7, 11, 5, 13, 8]$ و $n = 6$
  - راه حل: $[5, 7, 8, 10, 11, 13]$
- مساله B
  - نمونه: $S = [10, 7, 11, 5, 13, 8]$ و $n = 6$ و $x = 5$
  - راه حل: «بله، x در S وجود دارد»
- چنانچه الگوریتمی بتواند برای هر نمونه از ورودی، خروجی درست را تولید کند و متوقف شود، آن الگوریتم درست (correct) است.
- چناچه نرخ خطای الگوریتم های نادرست قابل کنترل باشد می توان از آنها نیز استفاده کرد.

مراحل حل یک مسأله

۱ – تعریف و بیان مسأله و فهم کامل آن

۲– تشخیص و تدوین یک مدل برای حل مسأله فوق (بررسی کلیه روش‌های حل مسأله)

۳– طراحی الگوریتم بر اساس مدل انتخاب شده

۴– بررسی و ارزیابی صحت الگوریتم

۵– تحلیل الگوریتم و ارزیابی پیچیدگی آن

۶– پیاده‌سازی الگوریتم با استفاده از زبان‌های برنامه‌سازی

۷– تست برنامه

۸– مستندسازی

روش های حل مساله :

- روش تقسیم و حل (Divide & Conquer)
- روش حریصانه (Greedy)
- روش برنامه‌نویسی پویا (Dynamic Programming)
- روش بازگشت به عقب (Back Tracking)
- روش شاخه و حد (Branch & Bound)

بیان الگوریتم

۱ – تشریح الگوریتم توسط یک زبان طبیعی (فارسی یا انگلیسی) برای الگوریتم‌های آسان و کوچک.

- یک الگوریتم برای مساله B

با شروع از اولین عنصر S، به ترتیب x را با هریک از عناصر S مقایسه کن تا اینکه x را پیدا کنی

و یا به آخر لیست S برسی. اگر x پیدا شد، پاسخ «بله» و در غیر این صورت پاسخ «خیر» را تولید کن.

- معایب نوشتن الگوریتم ها به زبان طبیعی
  - مشکل بودن نوشتن و درک الگوریتم های پیچیده
  - مشکل بودن ترجمه آن به یک زبان برنامه نویسی

۲– نمایش گرافیکی الگوریتم توسط فلوچارت (برای الگوریتم‌های آسان و کوچک)

۳– نمایش با استفاده از شبه کد (Pseudo Code) که شباهت زیادی به زبان‌های C و پاسکال دارد.

جستجوی ترتیبی

- الگوریتم ۱-۱ جستجوی ترتیبی :

مساله: آیا کلید x در آرایه S با n کلید قرار دارد؟ (اگر x در S نباشد خروجی صفر می باشد)

ورودی ها (پارامتر ها):

- عدد صحیح و مثبت n،
- آرایه S از کلیدها با اندیس ۱ تا n
- کلید x

خروجی ها:

- $Location$ (مکان x در S)

```
void seqsearch  ( int n,
                  const keytype S [ ],
                  keytype x,
                  index& location )
{
    location = 1 ;
    while  ( location <= n && S [location] != x)
        location++ ;
        if  ( location > n)
            location = 0 ;
}
```

تفاوت های شبه کد با C++

- نحوه استفاده از آرایه ها

- در C++ فقط مجاز به استفاده از اندیس های صحیح با شروع از صفر هستیم.
- در شبه کد هر جا که لازم باشد از آرایه هایی استفاده می کنیم که اندیس آنها در بازه دیگری از اعداد صحیح قرار می گیرد و یا اندیس آنها اصلا اعداد صحیح نیستند.
- در شبه کد طول آرایه ای دو بعدی را هنگام ارسال به زیر برنامه ها متغیر در نظر می گیریم. (مانند الگوریتم ۱-۴)
- در شبه کد می توانیم آرایه های محلی با طول متغیر تعریف کنیم. مثال:

```
void example ( int n )
{
    keytype S [2..n]
    ...
}
```

تفاوت های شبه کد با C++

- در شبه کد هر گاه بتوانیم، مراحل را با وضوح بیشتر با استفاده از روابط ریاضی و توضیحات انگلیسی نشان می دهیم. مثال:

if ($low \le x \le high$) { … }

exchange x and y;

- در شبه کد از انواع داده ای زیر که در C++ تعریف نشده اند استفاده می کنیم:

نوع داده ای	معنی
index	متغیر صحیحی که به عنوان اندیس به کار می رود.
number	متغیری که می توان آن را به عنوان عدد صحیح (int)و یا حقیقی (real) تعریف کرد
bool	متغیری که می تواند مقادیر true و false را بپذیرد
keytype	متغیری که مقدارش از یک مجموعه مرتب انتخاب می شود

- ساختار کنترلی غیر استاندارد:

repeat (n times) { …}

جمع نمودن عناصر آرایه

◄ الگوریتم ۱ – ۲ جمع نمودن عناصر آرایه

مساله: تمام اعداد موجود در آرایه n عنصری S را با هم جمع کنید.

ورودی ها:

- عدد صحیح و مثبت n،
- آرایه S با اندیس ۱ تا n.

خروجی ها:

- sum، (حاصل جمع اعداد موجود در S).

```
number sum( int n, const number S [ ] )
{
    index  i ;
    number  result ;

    result = 0 ;
    for  ( i = 1; i <= n; i++)
            result = result + S [i] ;
    return result ;
}
```

مرتب سازی تعویضی

◄ الگوریتم ۱-۳ مرتب سازی تعویضی

مساله: n کلید را به ترتیب غیر نزولی مرتب کنید.

ورودی ها:

- عدد صحیح و مثبت n،
- آرایه S از کلید ها با اندیس ۱ تا n.

خروجی ها:

- آرایه S حاوی کلیدها
- به ترتیب غیر نزولی.

```
void exchangesort ( int n, keytype S [ ] )
{
    index i, j ;
    for ( i = 1; i <= n - 1; i++)
        for ( j = i + 1; j <= n; j++)
            if ( S [j] < S [i] )
                exchange S [i], S [j];
}
```

ضرب ماتریس ها

◄ الگوریتم ۱ –۴ ضرب ماتریس ها

مساله: حاصل ضرب دو ماتریس n X n را تعیین کنید.

ورودی ها:

- عدد صحیح ومثبت n،
- آرایه های دو بعدی A و B
- A,Bبا اندیس ۱ تا n.

خروجی:

آرایه دو بعدی C از اعداد،

```
void matrixmult  ( int n, const number A [ ][ ],
                          const number B [ ][ ],
                          number C [ ][ ] )
{
    index  i, j, k ;
    for  ( i = 1; i <= n; i++)
        for  (j = 1; j <= n; j++)
            C [ i ] [ j ] = 0 ;
            for  ( k = 1; k <= n; k++)
                C [ i ] [ j ] = C [ i ] [ j ] + A [ i ] [ k ] * B [ k ] [ j ] ;
}
```

جستجوی دودویی

◄ الگوریتم ۱-۵ جستجوی دودویی

مساله: تعیین کنید آیا x در آرایه مرتب S با n کلید قرار دارد یا خیر.(اگر x در S نباشد صفر می باشد)

ورودی ها:

- عدد صحیح و مثبت n،
- آرایه مرتب S
- غیرنزولی با اندیس ۱ تا n
- کلید x

خروجی ها:

- $location$ (مکان x در S)

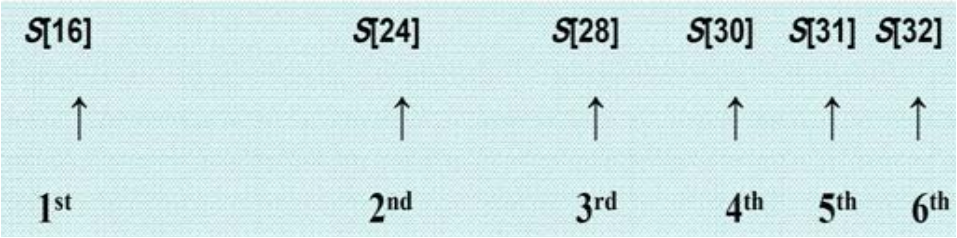
```
void  binsearch  ( int n, const keytype S [ ],  keytype x, index& location )
{
    index  low, high, mid ;

    low = 1 ;   high = n ;
    location = 0 ;
    while  ( low <= high && location == 0) {
        mid = ⌊ ( low + high ) / 2 ⌋ ;
        if  ( x == S [mid]
            location = mid ;
        else if  ( x < S [mid] )
            high = mid – 1 ;
        else
            low  = mid + 1 ;
    }
}
```

مقایسه دو الگوریتم جستجو

- با فرض داشتن آرایه ۳۲ عنصری

➢ Linear search: ۳۲ مقایسه در بدترین حالت

➢ Binary search:



اندازه آرایه	تعداد مقایسه های انجام شده توسط جستجوی دودویی (lg n +1)	تعداد مقایسه های انجام شده توسط جستجوی ترتیبی (n)
۱۲۸	۸	۱۲۸
۱۰۲۴	۱۱	۱۰۲۴
۱۰۴۸۵۷۶	۲۱	۱۰۴۸۵۷۶
۴۲۹۴۹۶۷۲۹۴	۳۳	۴۲۹۴۹۶۷۲۹۴

- به طور کلی
  - جستجوی ترتیبی: n
  - جستجوی دودویی (اگر n توانی از ۲ باشد): $\lg n + 1$

دنباله فیبونانچی

◄ الگوریتم ۱ – ۶ جمله n اُم فیبونانچی

مساله: جمله n اُم از دنباله فیبونانچی را تعیین کنید.

ورودی ها:

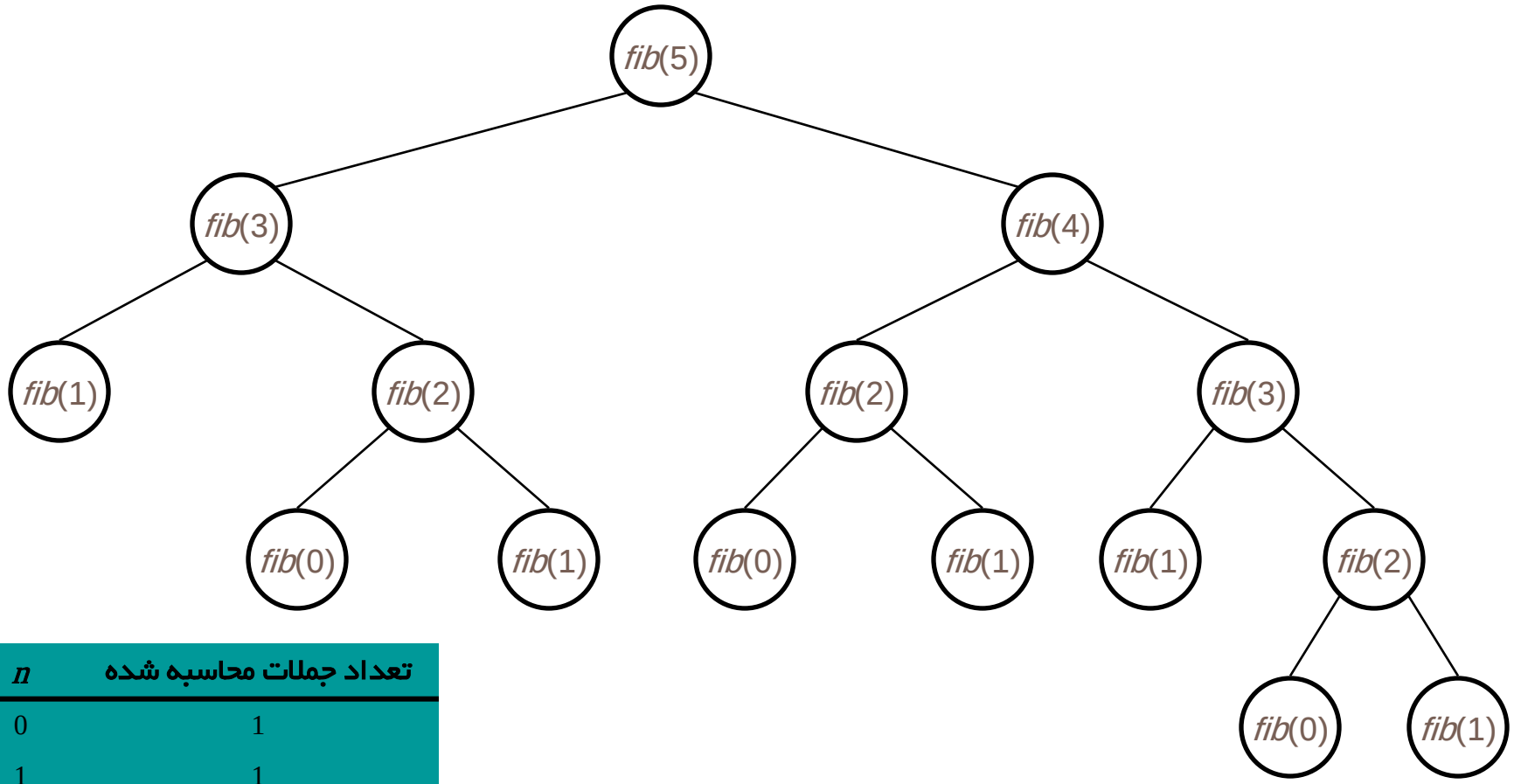
- یک عدد صحیح و غیر منفی n.

خروجی ها:

- جمله n اُم از دنباله فیبونانچی.

```
int fib ( int n)
{
    if ( n <= 1)
        return n ;
    else
        return fib ( n – 1) + fib ( n - 2) ;
}
```

الگوریتم جمله n اُم فیبوناچی (بازگشتی)



n	تعداد جملات محاسبه شده
0	1
1	1
2	3
3	5
4	9
5	15
6	25

- علت ناکارآیی: محاسبات تکراری
  - مثلا در این مثال $fib(2)$ سه بار محاسبه شده است.

تعداد جملات محاسبه شده

- هر بار که n به اندازه ۲ واحد افزایش می یابد، تعداد جملات محاسبه شده بیش از ۲ برابر افزایش می یابد، یعنی:

- $T(n) > 2 * T(n-2) > 2^{n/2} \qquad \text{when } n \geq 2$

- $T(n) > 2 * T(n-2)$

$$> 2 * 2 * T(n-4)$$

$$> 2 * 2 * 2 * T(n-6)$$

$$\ldots$$

$$> 2 * 2 * \ldots * 2 * T(0) = 2^{n/2}$$

- اثبات بوسیله استقراء

اثبات بوسیله استقراء

- پایه استقراء:

$$T(2) = 3 > 2 = 2^{2/2}$$

$$T(3) = 5 > 2.83 \approx 2^{3/2}$$

- فرض استقراء:

$$T(m) > 2^{m/2} \qquad , 2 \le m < n$$

- گام استقراء:

$$T(n) = T(n-1) + T(n-2) + 1$$

$$> 2^{(n-1)/2} + 2^{(n-2)/2} + 1$$

$$> 2^{(n-2)/2} + 2^{(n-2)/2} = 2 * 2^{(n-2)/2} = 2^{n/2}$$

محاسبه جمله nام فیبوناچی (تکراری)

◄ الگوریتم ۱–۷ جمله n اُم فیبونانچی (تکراری)

مساله: جمله n اُم از دنباله فیبونانچی را تعیین کنید.

ورودی ها:

یک عدد صحیح و غیر منفی n.

خروجی ها:

$fib2$،

جمله n اُم از دنباله فیبونانچی.

```
int fib2 ( int n)
{
    int f[0 . . n] ;
    f[0] = 0 ;
    if ( n > 0) {
        f[1] = 1 ;
        for ( i = 2; i <= n; i++)
            f[i] = f[i -1] + f [i – 2] ;
    }
    return f[n] ;
}
```

مقایسه دو الگوریتم فیبوناتچی

n	روش تکراری (n+1)	روش بازگشتی ($2^{n/2}$)	زمان اجرای تکراری	زمان اجرای بازگشتی
۴۰	۴۱	۱,۰۴۸۵۷۶	41ns	1048 µs
۶۰	۶۱	$1.1*10^{9}$	61ns	1 s
۱۰۰	۱۰۱	$1.1*10^{15}$	101ns	13 days
۱۲۰	۱۲۱	$1.2*10^{18}$	121ns	36 years
۱۶۰	۱۶۱	$1.2*10^{24}$	161ns	$3.8*10^{7}$ years
۲۰۰	۲۰۱	$1.3*10^{30}$	201ns	$4*10^{13}$ years

با فرض محاسبه هر جمله در مدت ۱ نانوثانیه ✓

Note: $1\ \mu s = 10^{-6}\ s$

$1\ ns = 10^{-9}\ s$

تحلیل الگوریتم ها

- هدف از تحلیل الگوریتم ها:
  - بررسی رفتار الگوریتم از نظر زمان اجرا و مقدار حافظه مصرفی قبل از پیاده سازی
  - مقایسه الگوریتم ها از نظر کارآیی

➢ Time complexity

➢ Space complexity

- عوامل موثر در زمان اجرای یک برنامه :
  - سرعت سخت افزار
  - نوع کامپایلر (بهینگی کد مقصد)
  - برنامه نویس (بهینگی کد منبع)
  - اندازه ورودی
  - ترکیب داده های ورودی
  - پیچیدگی الگوریتم
- مهمترین عامل، پیچیدگی الگوریتم می باشد که خود تابعی از اندازه ورودی می باشد. ترکیب داده های ورودی را نیز می توان با محاسبه پیچیدگی در حالت های مختلف در نظر گرفت.

- **پیچیدگی فضا (Space Complexity):**
  - میزان حافظه مورد نیاز از اجرا تا تکمیل الگوریتم می‌باشد.
  - به حاصل جمع فضای مورد نیاز متغیرها، آرایه ها ، پشته ها و کلیه ساختمان داده‌های مورد نیاز و همچنین فضای ذخیره کد برنامه در حافظه ، پیچیدگی فضای الگوریتم گفته می‌شود.
  - باید بر حسب n (تعداد ورودی‌ها) سنجیده شود.
  - پیچیدگی فضای کد زیر؟

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

  - O(1) می‌باشد.
- اگر پیچیدگی فضای یک الگوریتم به n وابسته نباشد، پیچیدگی آن را ثابت فرض کرده و از مرتبه O(1) می‌دانیم.

تحلیل الگوریتم ها

- تحلیل پیچیدگی زمان (Time Complexity):
  - محاسبه کارآیی بر حسب زمان
  - مستقل از کامپیوتر، زبان برنامه نویسی، برنامه نویس و تمامی جزئیات الگوریتم
  - محاسبه تعداد دفعات اجرای عملیات اصلی(مقایسه، جمع، ضرب و ...) بر حسب اندازه ورودی
- برای تحلیل پیچیدگی یک الگوریتم تابعی به نام $T(n)$ درنظر می گیریم که در آن n اندازه ورودی می باشد.
- در بسیاری از الگوریتم ها یافتن میزانی منطقی از اندازه ورودی آسان است، مثال:
  - الگوریتم ۱-۱ (جستجوی ترتیبی)
  - الگوریتم ۱-۲ (محاسبه مجموع عناصر آرایه)
  - الگوریتم ۱-۳ (مرتب سازی تعویضی)
  - الگوریتم ۱-۵ (جستجوی دودویی)

  ← اندازه ورودی: n، تعداد عناصر آرایه
  - الگوریتم ۱-۴ (ضرب ماتریس ها) ← اندازه ورودی: n، تعداد سطر ها و ستون ها
- در برخی از الگوریتم ها بهتر است اندازه ورودی را برحسب دو عدد بسنجیم:
  - اگر ورودی الگوریتم یک گراف باشد، ورودی را برحسب تعداد رئوس (n)وتعدادیالها (m) می سنجیم

عملیات اصلی

- عمل اصلی: دستور یا مجموعه ای از دستورات به طوری که کل کار انجام شده توسط الگوریتم، تقریبا متناسب با تعداد دفعات اجرای این دستور یا مجموعه دستورات باشد.
- مثال: الگوریتم ۱-۱ (جستجوی ترتیبی) و الگوریتم ۱-۵ (جستجوی دودویی)
  - در هر باز گذر از حلقه عنصر x با یک عنصر از S مقایسه می شود.
  - با تعیین اینکه هر یک از این الگوریتم ها چند بار این عمل اصلی را به ازای هر مقدار از n انجام می دهند، می توان کارآیی این دو الگوریتم را مقایسه نمود.
- تحلیل پیچیدگی زمانی:
  - تعیین تعداد دفعاتی که عمل اصلی به ازای هر مقدار از اندازه ورودی انجام می شود.
- انتخاب عمل اصلی بیشتر بر اساس تجربه و داوری انجام می شود.
- در برخی موارد ممکن است بخواهیم دو عمل اصلی متفاوت را در نظر بگیریم.
  - مرتب سازی: مقایسه و انتساب، هر یک به تنهایی می توانند عمل اصلی باشند.

تحلیل پیچیدگی در حالات مختلف

- در برخی موارد مانند الگوریتم ۱-۲ (جمع نمودن عناصر آرایه)، عمل اصلی همواره به ازای یک نمونه n به یک میزان انجام می شود. در چنین مواردی:

$T(n)$ = تعداد دفعاتی که الگوریتم عمل اصلی را بازاء یک نمونه n انجام می دهد

- در برخی موارد دیگر، تعداد دفعات اجرای عمل اصلی نه تنها به اندازه ورودی بلکه به مقادیر ورودی نیز بستگی دارد
  - مثال: جستجوی ترتیبی (با اندازه ورودی برابر n)
    - اگر x در اولین مکان آرایه باشد: تعداد مقایسه ها = ۱
    - اگر x در آرایه نباشد: تعداد مقایسه ها = n
- عمل اصلی همواره به ازای هر اندازه نمونه n، به تعداد دفعات یکسانی انجام می گیرد.

تحلیل پیچیدگی در حالات مختلف

- **بهترین حالت B(n)**
  - حالتی که الگوریتم می تواند سریعترین زمان اجرا را داشته باشد.
  - کران پایین زمان اجرا است.
- **بدترین حالت W(n)**
  - حالتی که زمان اجرای الگوریتم هرگز از آن بیشتر نخواهد شد.
  - کران بالای زمان اجرا است.
  - ***جهت مقایسه بین الگوریتم ها به طورمعمول از این تابع استفاده میشود.***
- **حالت میانگین A(n)**
  - برای تحلیل آن نیاز به در نظر گرفتن توزیع های مختلف مقادیر ورودی می باشد.
  - محاسبه آن مشکل است.

فرمول های سری

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

$$\sum_{i=1}^{N} C = C * N$$

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

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

$$\sum_{i=1}^{N} A^i = \frac{A^{N+1}-1}{A-1}, \quad \text{for some number } A$$

$$\sum_{i=1}^{N} i2^i = (N-1)2^{N+1}+2$$

$$\sum_{i=0}^{N} 2^i = 2^{N+1}-1$$

$$\sum_{i=1}^{N} C * i = C * \sum_{i=1}^{N} i, \text{ with } C \text{ a constant expression not dependent on } i$$

$$\sum_{i=C}^{N} i = \sum_{i=0}^{N-C} (i+C)$$

$$\sum_{i=C}^{N} i = \sum_{i=0}^{N} i - \sum_{i=0}^{C-1} i$$

$$\sum_{i=1}^{N} (A+B) = \sum_{i=1}^{N} A + \sum_{i=1}^{N} B$$

$$\sum_{i=0}^{N} (N-i) = \sum_{i=0}^{N} i$$

$$\sum_{i=1}^{N} \frac{1}{i} = \ln N$$

$$\sum_{i=1}^{N} \lg i \approx N \lg N - 1.5$$

فرمول های توابع لگاریتمی

$$\log_B 1 = 0$$

$$\log_B B = 1$$

$$\log_B(X * Y) = \log_B X + \log_B Y$$

$$\log_B X^Y = Y * \log_B X$$

$$\log_A X = \frac{(\log_B X)}{(\log_B A)}$$

For all real $a > 0$, $b > 0$, $c > 0$, and n,

$$\begin{aligned}
a &= b^{\log_b a}, \\
\log_c(ab) &= \log_c a + \log_c b, \\
\log_b a^n &= n \log_b a, \\
\log_b a &= \frac{\log_c a}{\log_c b}, \\
\log_b(1/a) &= -\log_b a, \\
\log_b a &= \frac{1}{\log_a b}, \\
a^{\log_b c} &= c^{\log_b a},
\end{aligned}$$

$$\begin{aligned}
\lg n &= \log_2 n \quad \text{(binary logarithm)}, \\
\ln n &= \log_e n \quad \text{(natural logarithm)} \\
\lg^k n &= (\lg n)^k \quad \text{(exponentiation)}, \\
\lg\lg n &= \lg(\lg n) \quad \text{(composition)}.
\end{aligned}$$

$$a^{\log_b^n} = n^{\log_b^a}$$

$$(\log n)^{\log n} = n^{\log\log n}$$

خصوصیات توابع نمایی(Exponential)

For all real $a > 0$, m, and n, we have the following identities:

$$
\begin{aligned}
a^0 &= 1\,, \\
a^1 &= a\,, \\
a^{-1} &= 1/a\,, \\
(a^m)^n &= a^{mn}\,, \\
(a^m)^n &= (a^n)^m\,, \\
a^m a^n &= a^{m+n}\,.
\end{aligned}
$$

تحلیل پیچیدگی زمانی الگوریتم جستجوی خطی

- جستجوی خطی (linear search)
  - عمل اصلی: تعداد مقایسه
  - اندازه ورودی: تعداد عناصر آرایه n
  - بهترین حالت: وقتی که عنصر مورد نظر در اولین خانه باشد $B(n)=1$
  - بدترین حالت: وقتی که عنصر مورد نظر در آرایه موجود نباشد $W(n)=n$
  - حالت متوسط:
    - حالت ۱) فرض می کنیم که x در S وجود دارد
      - عناصر S همگی متمایز می باشند
      - احتمال وجود x در همه مکانهای آرایه یکسان است $(1/n)$

$$A(n)=\sum_{k=1}^{n}\left(k\times\frac{1}{n}\right)=\frac{1}{n}\sum_{k=1}^{n}k=\frac{1}{n}\times\frac{n(n+1)}{2}=\frac{n+1}{2}$$

تحلیل پیچیدگی زمانی الگوریتم جستجوی خطی

- **حالت ۲)** ممکن است که x در S نباشد
  - فرض می کنیم که احتمال وجود x در S برابر p باشد
  - عناصر S همگی متمایز می باشند
  - به شرط وجود x در S، احتمال وجود x در همه مکانهای آرایه یکسان است (p/n)

$$A(n)=\sum_{k=1}^{n}\left(k\times\frac{p}{n}\right)+n(1-p)=\frac{p}{n}\times\frac{n(n+1)}{2}+n(1-p)=n\left(1-\frac{p}{2}\right)+\frac{p}{2}$$

تحلیل پیچیدگی زمانی

تحلیل پیچیدگی زمانی الگوریتم مرتب سازی تعویضی :

- عمل اصلی: مقایسه $S[j]$ و $S[i]$
- اندازه ورودی: n، تعداد عناصر آرایه که باید مرتب شوند.
- تحلیل پیچیدگی:

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} n - i$$

$$= (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$$

تحلیل پیچیدگی زمانی الگوریتم ضرب ماتریس ها:

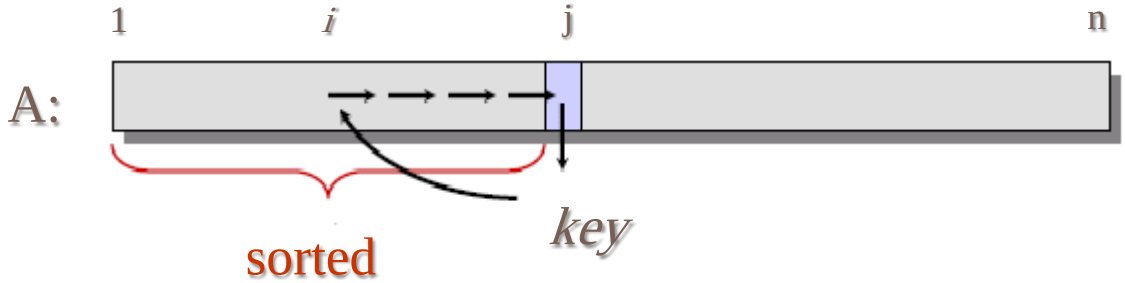
- عمل اصلی: دستور ضرب در داخلی ترین حلقه for
- اندازه ورودی: n، تعداد سطرها و ستون ها
- تحلیل پیچیدگی:

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} 1 = \sum_{i=1}^{n} \sum_{j=1}^{n} n = \sum_{i=1}^{n} n^2 = n^3$$

تحلیل پیچیدگی زمانی الگوریتم مرتب سازی درجی

```
INSERTION-SORT (A, n)        ⊳ A[1 . . n]
c1:     for j ←2 to n
        {
          //Insert a[j] into the sorted sequence a[1..j-1]
c3:     do key ← A[ j]
c4:      i ← j –1
c5:      while i > 0 and A[i] > key
           {
c6:          do A[i+1] ← A[i]
c7           i ← i –1
           }
c8:        A[i+1] = key
        }
```

هزینه	دفعات اجرا
-	-
c_1	n
-	-
0	n - 1
c_3	n - 1
c_4	n - 1
c_5	$\sum_{j=2}^{n} t_j$
-	-
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$\sum_{j=2}^{n} (t_j - 1)$
-	-
c_8	n -1
-	-
-	-



مثالی از مرتب سازی درجی

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *done*

پیچیدگی زمانی مرتب سازی درجی

کل زمان اجرای مرتب‌سازی درجی

$$T(n)=c_1n+c_3(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n}t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8(n-1)$$

- بهترین حالت(Best Case) :
- زمانی است که آرایه از قبل مرتب باشد. tj=1

$$T(n)=c_1n+c_3(n-1)+c_4(n-1)+c_5(n-1)+c_8(n-1)$$
$$=(c_1+c_3+c_4+c_5+c_8)n-(c_3+c_4+c_5+c_8)$$

- هزینه زمانی مرتب سازی درجی در بهترین حالت، تابعی خطی و به صورت an+b می باشد.

پیچیدگی زمانی مرتب سازی درجی

کل زمان اجرای مرتب‌سازی درجی

$$T(n)=c_1n+c_3(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n}t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8(n-1)$$

- **بدترین حالت(Worst Case):**
- زمانی است که آرایه از قبل به ترتیب عکس مرتب باشد.
- در این صورت هر عنصر a[j] باید با تمامی عناصر مرتب شده در زیر آرایه a[1..j-1] مقایسه شود.
- بنابراین برای $j=2,3,\ldots,n$ داریم $t_j=j$

$$T(n)=c_1n+c_3(n-1)+c_4(n-1)+c_5(\frac{n(n+1)}{2}-1)+c_6(\frac{n(n-1)}{2})+c_7(\frac{n(n-1)}{2})+c_8(n-1)$$

$$=(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2})n^2+(c_1+c_3+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8)n-(c_3+c_4+c_5+c_8)$$

- هزینه زمانی مرتب سازی درجی در بدترین حالت، تابعی درجه دوم به صورت an^2+bn+c می باشد.

پیچیدگی زمانی مرتب سازی درجی

کل زمان اجرای مرتب‌سازی درجی

$$T(n)=c_1n+c_3(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n}t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8(n-1)$$

- **حالت متوسط(Average Case):**
- می توان فرض کرد a[j] از نصف عناصر a[1..j-1] کوچکتر است
- در این صورت هر عنصر a[i] باید با نصف عناصر مرتب شده در زیر آرایه a[1..j-1] مقایسه شود.
- بنابراین برای $j=2,3,\ldots,n$ داریم $t_j=j/2$
- هزینه زمانی مرتب سازی درجی در بدترین حالت، تابعی درجه دوم به صورت an^2+bn+c می باشد.
- حالت متوسط در این الگوریتم همانند بدترین حالت می باشد.

تمرین

- تمرین در کلاس– زمان اجرای الگوریتم جمع دو ماتریس را محاسبه کنید.
- الگوریتم های مرتب سازی را مطالعه نمائید.

- **تمرین**

۱ – زمان اجرای الگوریتم ضرب ماتریس ها را تحلیل کنید.

۲– الگوریتمی بنویسید که دومین بزرگترین عنصر یک لیست n عنصری را پیدا کند. در بدترین حالت چند مقایسه لازم است؟

۳–الگوریتم مرتب سازی انتخابی(selection sort) با انتخاب کوچکترین عنصر از لیست و جایگزینی آن با عنصر اول، سپس پیدا کردن دومین کوچکترین عنصر و جایگزینی آن با عنصر دوم و تکرار این روش برای n-1 عنصر اول لیست عمل می کند. شبه کد الگوریتم را بنویسید و بهترین حالت و بدترین حالت زمان اجرای آن را محاسبه کنید.

مرتبه رشد(نرخ رشد)

- مرتبه یا نرخ رشد زمان اجرای الگوریتم :
  - وقتی مقدار ورودی(n) بزرگ باشد، بیان دقیق زمان اجرا برحسب n ضروری نیست زیرا جمله با بزرگترین درجه در زمان موثر است.

N	$n^۲$	$n^۲ + n$
۱	۱	۲
۱۵	۲۵	۳۰
۱۰	۱۰۰	۱۱۰
۱۰۰	۱۰۰۰۰	۱۰۱۰۰
۱۰۰۰	۱۰۰۰۰۰۰	۱۰۰۱۰۰۰

مرتبه رشد(نرخ رشد)

- مرتبه یک الگوریتم با بزرگترین جمله تابع پیچیدگی زمانی آن تعیین می شود.
- پیچیدگی و مرتبه اجرایی الگوریتم به n وابسته است و تابعی از n می‌باشد و آنرا با $T(n)$نشان می‌دهیم.
- الگوریتم هایی با پیچیدگی زمانی ازقبیل n ، 100n را الگوریتم های زمانی خطی می گویند.
- مجموعه کامل توابع پیچیدگی را که با توابع درجه دوم محض قابل دسته بندی باشند، $\theta(n^2)$ می گویند.
- تابعی که عضو مجموعه $\theta(n^2)$ باشد، از مرتبه n2 است.
- مجموعه ای از توابع پیچیدگی که با توابع درجه سوم محض قابل دسته بندی باشند، **$\theta(n^3)$** نامیده می شوند.
- گروه های پیچیدگی:

$$\theta(\log n), \theta(n), \theta(n \log n), \theta(n^{۲}), \theta(n^{۳}), \theta(۲^{n}), \ldots$$

مرتبه رشد(نرخ رشد)

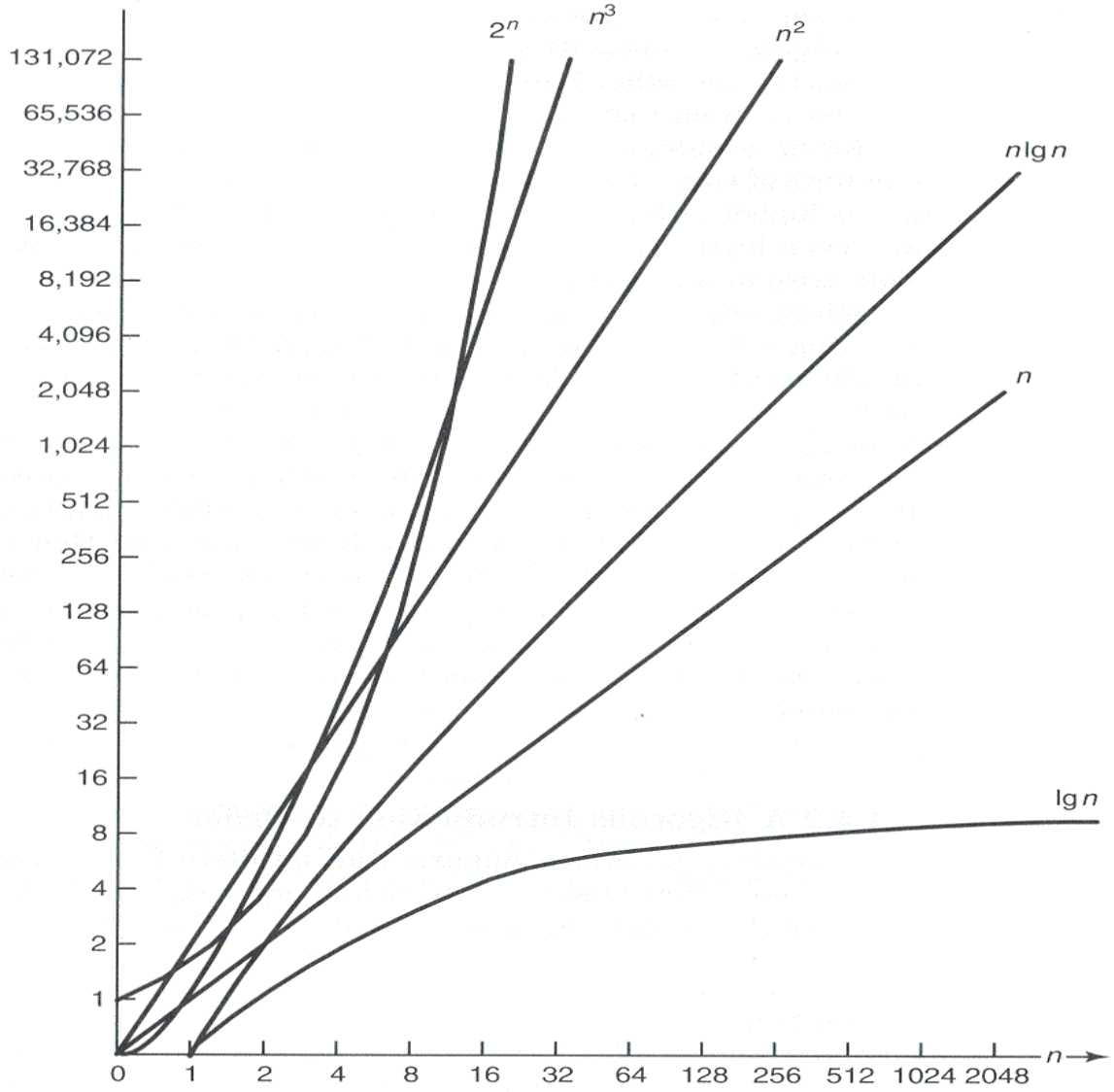


Figure 1.3 ● Growth rates of some common complexity functions.

مرتبه رشد(نرخ رشد)

- Table 1.4 Execution times for algorithms with the given time complexities

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 μs = 10^{-6} second.

†1 ms = 10^{-3} second.

نمادهای مجانبی(asymptotic notation)

$$O \quad , o \quad , \Omega \quad , \omega \quad , \theta$$

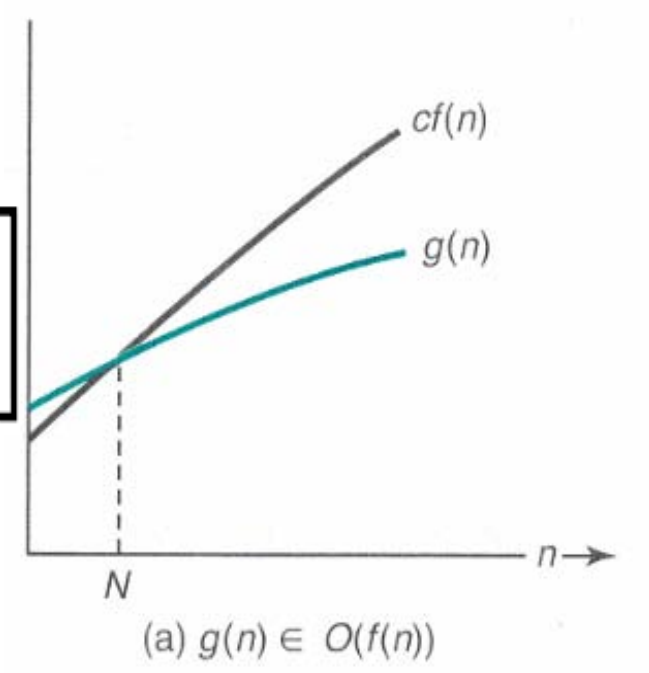
- برای توصیف زمان اجرای الگوریتم از نماد های مجانبی استفاده می کنیم.
- جهت مقایسه الگوریتم‌های مختلف با یکدیگر از نمادهای مجانبی استفاده می کنیم.

مرتبه اُی بزرگ (Big O)

- Big O (اُی بزرگ):
  - O (*f* (n)) مجموعه ای از توابع پیچیدگی g(n) است که برای آن ها یک ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی n >= N داریم:

$$g(n) <= c \times f(n)$$

$$g(n) \in O(f(n)) \Leftrightarrow c, N > 0 : \forall n \geq N \;\; g(n) \leq cf(n)$$



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

مرتبه اُی بزرگ (Big O)

- Big O (اُی بزرگ):

$$g(n) <= c \times f(n)$$

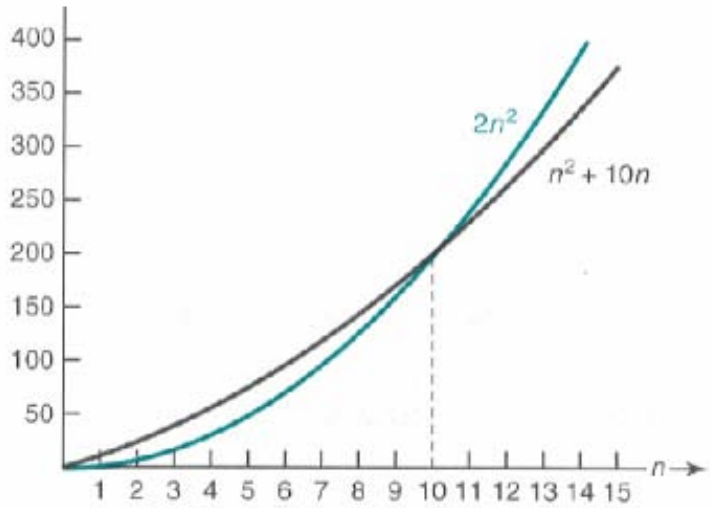


Figure 1.5 • The function $n^2 + 10n$ eventually stays beneath the function $2n$

- اُی بزرگ یک حد بالای مجانبی (کران بالا) بر روی یک تابع قرار می دهد.
- معنای شهودی اُی بزرگ: $g(n)$ حداقل به خوبی $f(n)$ می باشد.

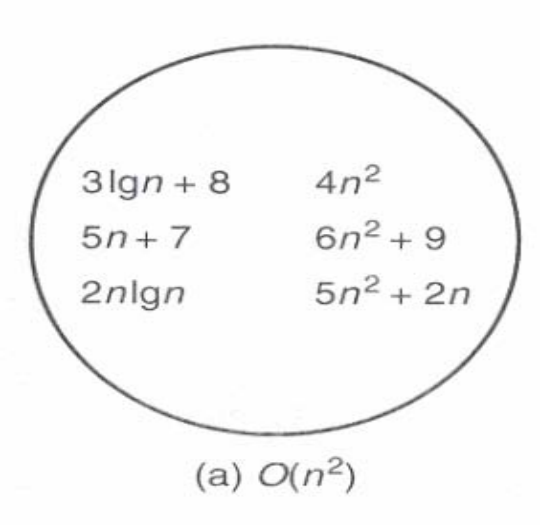
مرتبه اُی بزرگ (Big O)

- $5n^2 \in O(n^2)$
  $5n^2 \le 5n^2 \Rightarrow N = 0, c = 5$

- $T(n) = n(n-1)/2 \in O(n^2)$
  $n(n-1)/2 \le n(n)/2 = (1/2)n^2$
  $\Rightarrow N = 0, c = 1/2$

- $n^2 \in O(n^2 + 10n)$
  $N = 10, c = 1$

- $n \in O(n^2)$
  $N = 1, c = 1$



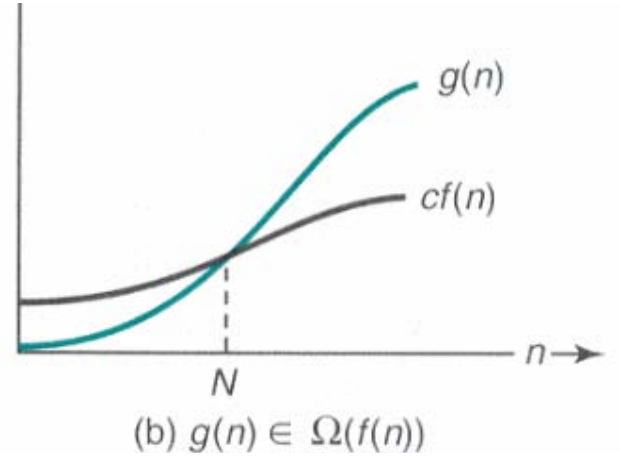
(a) $O(n^2)$

- به طور کلی $O(n^2)$ شامل تمام توابعی است که رشدشان کمتر یا مساوی n^2 است.

مرتبه اُمگای بزرگ $\Omega\ (f(n))$

- **Ω یا امگای بزرگ:**

- برای یک تابع پیچیدگی مفروض $f(n)$ ، $\Omega\ (f(n))$ مجموعه ای از توابع پیچیدگی $g\ (n)$ است که برای آن ها یک عدد ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد طوری که به ازای همه ی N <= n داریم:



$$g\ (n) >= c \times f\ (n)$$

- اگر $g(n)\ \varepsilon\ \Omega(f(n))$ می گوئیم $g(n)$ از امگای $f(n)$ می باشد.

- Ωیک حد پائین مجانبی(کران پایین حدی) بر روی یک تابع قرار می دهد.

- **تعریف شهودی:** $g(n)$ حداقل به بدی $f(n)$ است.

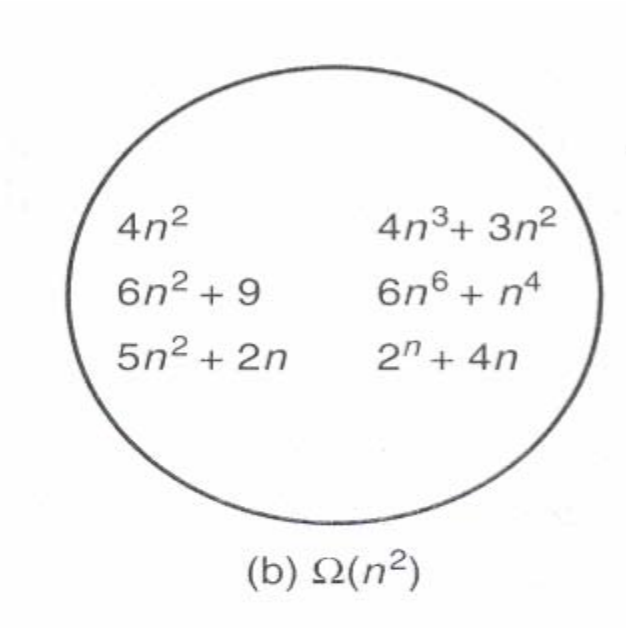
مرتبه اُمگای بزرگ $\Omega(f(n))$

- $5n^2 \in \Omega(n^2)$
  $5n^2 \geq 1 \times n^2 \Rightarrow N = 0,\ c = 1$

- $n^2 + 10n \in \Omega(n^2)$
  $n^2 + 10n \geq n^2 \Rightarrow N = 0,\ c = 1$

- $T(n) = n(n-1)/2 \in \Omega(n^2)$
  $n \geq 2 \Rightarrow n - 1 \geq n/2 \Rightarrow$
  $n(n-1)/2 \geq (n/2)(n/2) = n^2/4$
  $\Rightarrow N = 2,\ c = 1/4$

- $n^3 \in \Omega(n^2)$



- به طور کلی $\Omega(n^2)$ شامل تمام توابعی است که رشدشان بیشتریا مساوی n^2 است.

مرتبه $\Theta(f(n))$

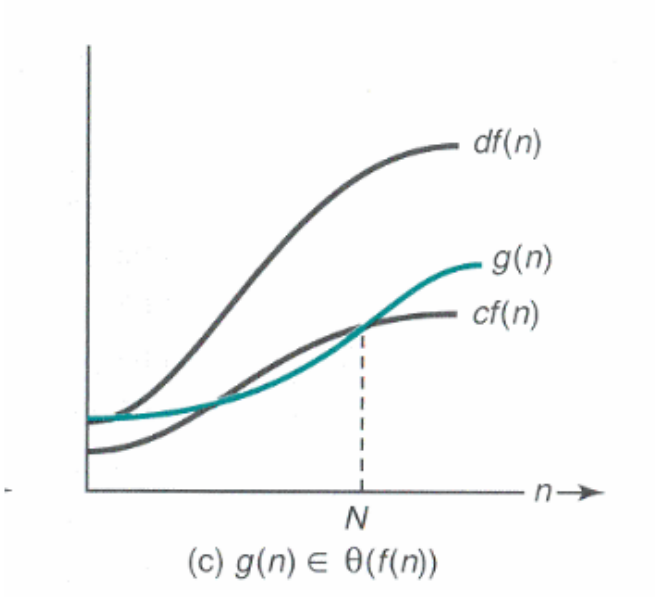
- برای یک تابع پیچیدگی مفروض $f(n)$، داریم:

$$\theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

- یعنی $\theta(f(n))$ مجموعه ای از توابع پیچیدگی $g(n)$ است که برای آن ها ثابت های حقیقی مثبت c و d و عدد صحیح غیر منفی N وجود دارد طوری که:

$$c \times f(n) <= g(n) <= d \times f(n)$$

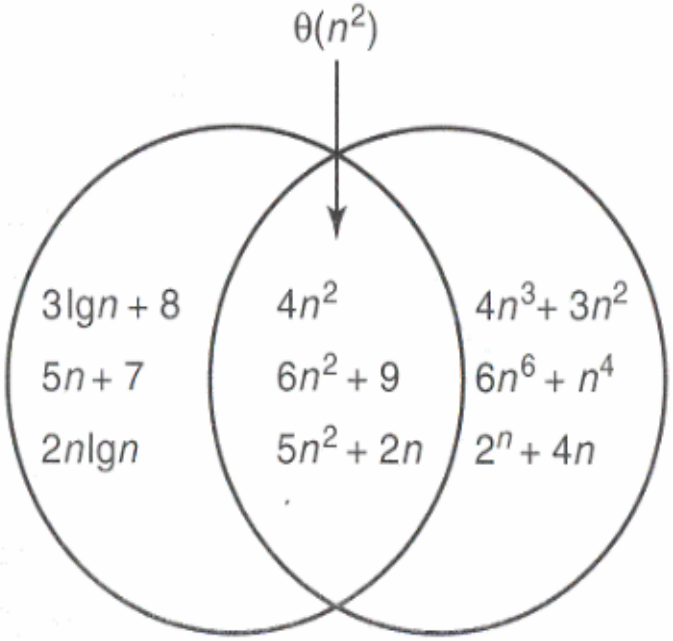
- تابع را از بالا و پایین محدود می کند.



(c) $g(n) \in \theta(f(n))$

مرتبه $\Theta(f(n))$

- $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- اگر $g(n) \in \Theta(f(n))$ ، می گوییم $g(n)$ از مرتبه (دقیق) $f(n)$ می باشد.



(c) $\theta(n^2) = O(n^2) \cap \Omega(n^2)$

اُی کوچک o(f(n))

اُی کوچک(Small o) : o(f(n))

برای یک تابع پیچیدگی $f(n)$ مفروض، $o(f(n))$ "o کوچک" عبارت است ازمجموعه کلیه توابع پیچیدگی$g(n)$ است که **به ازای هر** ثابت حقیقی مثبت c ،یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای N =< n داریم:

$$g(n) < c \times f(n)$$

اُمگای کوچک (Small ω) : ω(f(n))

برای یک تابع پیچیدگی $f(n)$ مفروض، $\omega(f(n))$ عبارت است از مجموعه کلیه توابع پیچیدگی $g(n)$ است که **به ازای هر ثابت حقیقی** مثبت c، یک عدد صحیح غیر منفی N وجود دارد طوری که به ازای N =< n داریم:

$$c \times f(n) < g(n)$$

o(f(n)) ‘ی کوچک

- $n \in o(n^2)$

فرض می کنیم $c > 0$ است. باید یک n بیابیم که برای هر $n \geq N$ داشته باشیم:

$$n \leq cn^2 \Rightarrow 1/c \leq n$$

بنابراین کافی است که هر $N \geq 1/c$ را انتخاب کنیم. مثلا اگر $c = 0.01$ باشد، باید

N بزرگتر مساوی ۱۰۰ باشد.

- $n \notin o(5n)$ (proof by contradiction)

فرض می کنیم $c = 1/6$ باشد. اگر $n \in o(5n)$ باشد، آنگاه باید یک Nوجود داشته باشد

که برای هر $n \geq N$ داشته باشیم: $n \leq (1/6)5n = (5/6)n$

و این تناقض ثابت می کند که $n \notin o(5n)$

مقایسه نمادهای مجانبی

- برای درک سریع نمادهای پیچیدگی می‌توان آنها را به صورت زیر با عملگرهای مقایسه‌ای هم‌ارز دانست :

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

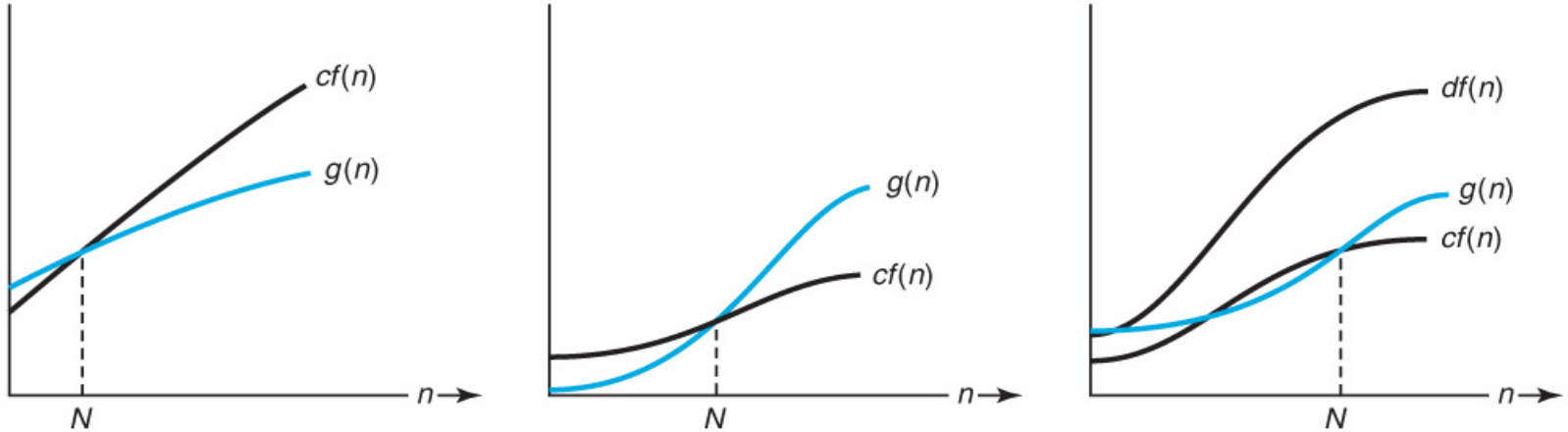
$$g(n) \in \Omega(f(n)) \rightarrow g(n) \geq f(n)$$

$$g(n) \in \theta(f(n)) \rightarrow g(n) = f(n)$$

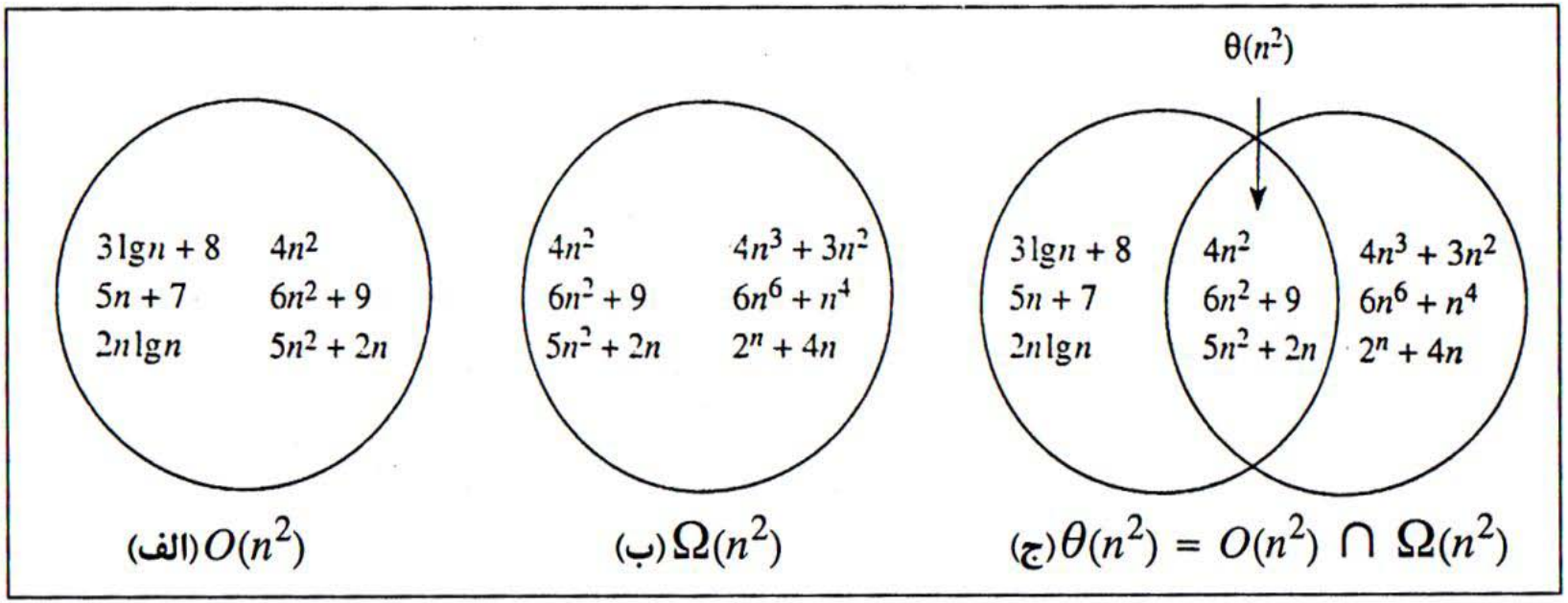
$$g(n) \in o(f(n)) \rightarrow g(n) < f(n)$$

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

مقایسه نمادهای مجانبی



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



(الف) $O(n^2)$ (ب) $\Omega(n^2)$ (ج) $\theta(n^2) = O(n^2) \cap \Omega(n^2)$

مقایسه نمادهای مجانبی

- عبارات زیر همگی درست هستند.

۱) $5n^2 - 6n = \theta(n^2)$

۲) $n! = O(n^n)$

۳) $2n^2 2^n + n\log n = \theta(n^2 2^n)$

۴) $\sum_{i=0}^{n} i^2 = \theta(n^3)$

۵) $\sum_{i=0}^{n} i^3 = \theta(n^4)$

۶) $n^{2^n} + 6\times 2^n = \theta(n^{2^n})$

۷) $n^3 + 10^6 n^2 = \theta(n^3)$

۸) $6n^3/(\log n + 1) = O(n^3)$

۹) $n^{1.001} + n\log n = \theta(n^{1.001})$

۱۰) $10n^3 + 15n^4 + 100n^2 2^n = O(n^2 2^n)$

۱۱) $33n^3 + 4n^2 = \Omega(n^2)$

۱۲) $33n^3 + 4n^2 = \Omega(n^3)$

۱۳) $6n^2 + 20n \in O(n^3)$

خصوصیات مرتبه

- $g(n) \in O(f(n)) \Leftrightarrow f(n) \in \Omega(g(n))$
- $g(n) \in \Theta(f(n)) \Leftrightarrow f(n) \in \Theta(g(n))$
- اگر $a > 1$ و $b > 1$ ، آنگاه $\log_a n \in \Theta(\log_b n)$

(یعنی تمامی توابع لگاریتمی در یک دسته پیچیدگی قرار می گیرند)

- اگر $b > a > 0$، آنگاه $a^n \in o(b^n)$ (یعنی توابع نمایی در یک دسته پیچیدگی قرار ندارند)
- برای هر $a > 0$، داریم $a^n \in o(n!)$ (یعنی $n!$ از هر تابع نمایی بدتر است)

خصوصیات مرتبه

- ترتیب زیر را از دسته های پیچیدگی مختلف در نظر بگیرید:

$$\Theta(1), \Theta(\lg n), \Theta(n), \Theta(n\lg n), \Theta(n^2), \Theta(n^j), \Theta(n^k), \Theta(a^n), \Theta(b^n), \Theta(n!), \Theta(n^n)$$

که در آن $k > j > 2$ و $b > a > 1$.

- اگر $g(n)$ در یک دسته پیچیدگی واقع در سمت چپ دسته پیچیدگی $f(n)$ باشد، آنگاه

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

- اگر $c \geq 0$، $d > 0$، $g(n) \in O(f(n))$ و $h(n) \in \Theta(f(n))$ آنگاه :

$$c \times g(n) + d \times h(n) \in \Theta\,(f(n))$$

استفاده از خصوصیات مرتبه

- تمام توابع لگاریتمی در یک دسته پیچیدگی قرار می گیرند. (ویژگی ۳)

$$\Theta(\log_4 n) = \Theta(\lg n)$$

- هر تابع لگاریتمی در نهایت بهتر از هر تابع چند جمله ای، هر تابع چند جمله ای در نهایت بهتر از هر تابع نمایی و هر تابع نمایی در نهایت بهتر از هر تابع فاکتوریل می باشد.

$$\lg n \in o(n) \qquad n^{10} \in o(2^n) \qquad 2^n \in o(n!)$$

- با اعمال دو ویژگی آخر به طور مکرر داریم:

$$5n + 3\lg n + 10n \lg n + 7n^2 \in \Theta(n^2)$$

یعنی در تعیین مرتبه، همواره اجازه حذف جملاتی از مرتبه پایین را داریم.

استفاده از حد برای تعیین مرتبه زمانی

اگر دو الگوریتم با پیچیدگی‌های $f(n)$ و $g(n)$ داشته باشیم برای مقایسه آنها می‌توان به صورت زیر از حد استفاده نمود:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases} c & : \quad g(n)\in\theta(f(n)) \\ 0 & : \quad g(n)\in\Omega(f(n)) \\ \infty & : \quad g(n)\in O(f(n)) \end{cases}$$

و یا به صورت زیر :

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases} c & : \quad f(n)\in\theta(g(n)) \\ 0 & : \quad f(n)\in O(g(n)) \\ \infty & : \quad f(n)\in\Omega(g(n)) \end{cases}$$

ترتیب رشد توابع

گوئیم رشد تابع f(n) از g(n) بیشتر است در صورتی که اگر n به سمت بینهایت میل کند آنگاه f(n) زودتر به بینهایت میل کند.

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n\log n) < O(n^2) < O(n^2 \log n) <$$

$$O(n^3) < O(2^n) < O(3^n) < O(n!) < O(n^n)$$

عبارت ریاضی	نسبت سرعت رشد
g(n)=O(f(n))	سرعت رشد g(n) ≤ سرعت رشد f(n)
g(n)=Ω(f(n))	سرعت رشد g(n) ≥ سرعت رشد f(n)
g(n)=θ(f(n))	سرعت رشد g(n) = سرعت رشد f(n)

مثال

- در رشد توابع زیر کدام ترتیب صحیح می‌باشد؟ (علوم کامپیوتر – دولتی ۸۵)

1- $O(n \log n)$, $O(1+\varepsilon)^n$, $O(\frac{n^2}{\log n})$

2- $O(1+\varepsilon)^n$, $O(n \log n)$, $O(\frac{n^2}{\log n})$

3- $O(\frac{n^2}{\log n})$, $O(n \log n)$, $O(1+\varepsilon)^n$

4- $O(n \log n)$, $O(\frac{n^2}{\log n})$, $O(1+\varepsilon)^n$

حل : گزینه ۱ و ۲ نادرست است.(چرا؟)

$$\lim_{n\to\infty}\frac{\frac{n^2}{\log n}}{n\log n}=\lim_{n\to\infty}\frac{n}{\log^2 n}\xrightarrow{\text{لوپیتال}}\lim_{n\to\infty}\frac{1}{2\times\log n\times\frac{1}{n}}=\lim_{n\to\infty}\frac{n}{2\log n}=\infty$$

$$\longrightarrow n\log n\in O(\frac{n^2}{\log n})$$

بنابراین گزینه ۴ صحیح است.

مثال

کدام عبارت صحیح است؟ (علوم کامپیوتر – دولتی۸۲)

1) $(n+1)(n^2-2n+1) \in O(2^n)$

2) $(n+1)(n^2-2n+1) \in \theta(n)$

3) $(n+1)(n^2-2n+1) \in \Omega(n^4)$

4) $(n+1)(n^2-2n+1) \in O(n^2 \log n)$

جواب:

$$\left.\begin{array}{c}(n+1)(n^2-2n+1) \in \theta(n^3) \\ n^3 \in O(2^n)\end{array}\right\} \Rightarrow (n+1)(n^2-2n+1) \in O(2^n)$$

بنابراین گزینه ۱ صحیح می‌باشد.

مثال

2- توابع $f(n) = 4^{\log n}$ ، $g(n) = (\log n)^{\log n}$ و $h(n) = \log^2 n$ را در نظر بگیرید. کدام یک از گزاره‌های زیر صحیح است؟ (مهندسی کامپیوتر – دولتی 85)

1) $f(n) \in O(g(n))\,,\ f(n) \in \Omega(h(n))$

2) $g(n) \in \Omega(h(n))\,,\ h(n) \in \Omega(f(n))$

3) $f(n) \in O(h(n))\,,\ g(n) \in \Omega(f(n))$

4) $h(n) \in O(g(n))\,,\ f(n) \in \theta(g(n))$

حل :

$$\left.\begin{array}{l} \boxed{\begin{array}{l} f(n) = 4^{\log n} = n^{\log 4} = n^2 \\ g(n) = (\log n)^{\log n} = n^{\log \log n} \\ h(n) = \log^2 n \end{array}} \\ h(n) \in o(f(n)) \Rightarrow f(n) \in \Omega(h(n)) \end{array}\right\} \Rightarrow \quad h(n) \in o(f(n)) \in o(g(n))$$

بنابراین گزینه ۱ صحیح می‌باشد.

نکته : از فرمولهای لگاریتمی زیر استفاده شده است :

$$(\log n)^{\log n} = n^{\log \log n} \qquad a^{\log_b^n} = n^{\log_b^a}$$

مثال

کدام یک از روابط ذیل درست است؟ (مهندسی کامپیوتر – آزاد ۸۱)

1) $O(\log n) < O(\sqrt{n})$

2) $O(\log n) > O(\sqrt{n})$

3) $O(n!) < O(a^n)$

4) $O(\sqrt{n^3}) < O(n)$

جواب: مقایسه های صحیح گزینه‌های فوق عبارتند از :

$$O(\log n) < O(\sqrt{n}) \quad \text{و} \quad O(n!) > O(a^n) \quad \text{و} \quad O(\sqrt{n^3}) > O(n)$$

بنابراین گزینه ۱ صحیح می‌باشد.

تمرین در کلاس

- توابع زیر را از نظر مرتبه رشد مرتب کنید.

2^n	$\lg \lg n$	$n^3 + \lg n$
$\lg n$	$n - n^2 + 5n^3$	2^{n-1}
n^2	n^3	$n \lg n$
$(\lg n)^2$	$\sqrt{n}$	6
$n!$	n	$(3/2)^n$

تمرین

۴–نمادهای مجانبی را از نظر خصوصیات بازتابی(reflexivity)، تقارن(symmetry)، تعدی(transitivity) و ترانهاده تقارنی (transpose symmetry) بررسی کنید.

۵– با استفاده از تعریف نمادها موارد زیر را ثابت کنید؟

الف) $\log(n) = O(n)$ ب) $6n^2+20n = \Theta(n^2)$

ج) $2^n = \Omega(5^{\log n})$

تمرین

۶– مقایسه زمان اجرا

برای هر کدام از توابع f(n) و زمان اجرای داده شده در جدول زیر، بزرگترین اندازه n را که مساله می تواند در زمان t حل شود را مشخص کنید. فرض کنید الگوریتم برای حل مساله به اندازه f(n) میکروثانیه زمان می برد.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

تمرین

۷–فرض کنید مرتب سازی درجی در $8n^2$ و مرتب سازی ادغام در $64n\log n$ یک ارایه n عنصری را مرتب می کند. برای چه مقادیری از n مرتب سازی درجی سریعتر است.

۸–کوچکترین مقدار n را به دست آورید که به ازای آن الگوریتمی که زمانش $100n^2$ است از الگوریتمی که زمانش 2^n است می باشد سریعتر است.

۹–یک الگوریتم مرتبه $\Theta(n\log n)$ ارائه دهید که یک عدد x و یک آرایه n عنصری S را دریافت کند و مشخص کند ایا هیچ دو عنصری از S وجود دارند که جمعشان S شود.

تمرین

۱۰ –زمان اجرای الگوریتم های مرتب سازی یک آرایه n عنصری را در بهترین حالت، حالت متوسط و بدترین حالت در یک جدول بنویسید.

روش مرتب سازی	Insertion	selection	Bubble	merge	quick	heap	
بهترین حالت							
حالت متوسط							
بدترین حالت							