موسسه آموز ش عالی ماد

تحلیل و طراحی الگوریتم ها

فصل اول (قسمت اول):

کارایی ، تحلیل و مرتبه الگوریتم ها، نمادهای مجانبی

سید جبار بارخدا
barkhoda@gmail.com

مهر ۹۲

رئوس مطالب درس

- **ارزیابی کارایی، تحلیل و مرتبه الگوریتم ها**
  - (Efficiency, Analysis, and Order)
    - مقدمه ای بر تحلیل الگوریتم ها
    - مرتبه الگوریتم ها
    - نماد های مجانبی

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.

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

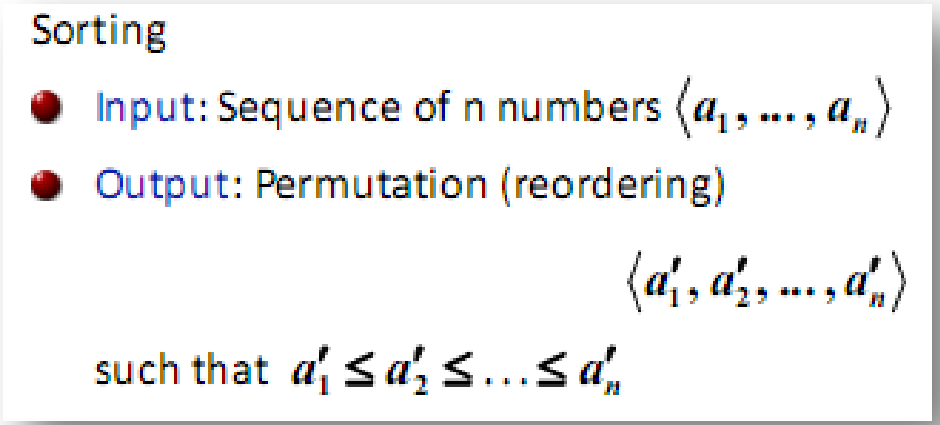
مطالعه الگوریتم ها

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.
  - مرتب سازی یک لیست S متشکل از n عدد به ترتیب غیر نزولی.
    - پاسخ دنباله مرتب از n عدد می باشد.



پارامترهای مسئله

- 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
    - نمونه مسئله ۱: $n = ۶ , S = \{۸ , ۱۳, ۵, ۱۱, ۷, ۱۰\}$
    - راه حل: $\{۵, ۷, ۸, ۱۰, ۱۱,۱۳\}$

الگوریتم درست/نادرست

- 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.
- به عنوان مثال عبارتی مانند "n را بگیر و در c ضرب 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.

شبه کد

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

if (low ≤ x ≤ high) {
...
}
exchange x and y ;

```
1.  void seqSearch (int n, const keyType s[], keyType x, index& location)
2.  {
3.      location = 0;
4.      while(location < n && s[location] != x)
5.          location++;
6.      if(location > n)
7.          location = -1;
8.  }
```

- برای هر الگوریتم ورودی و خروجی را تعیین کنید؟
  - الگوریتم جمع عناصر یک آرایه
  - الگوریتم ضرب ماتریسها
  - الگوریتم مرتب سازی عناصر به ترتیب غیر نزولی (*exchange sort- مرتب سازی تعویضی*)

الگوریتم جمع عناصر یک آرایه

```
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;
}
```

الگوریتم ضرب ماتریسها

```
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 ] ;
}
```

- الگوریتم مرتب سازی عناصر به ترتیب غیر نزولی (*exchange sort*- مرتب سازی تعویضی)

```
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] and S[j];
}
```

الگوریتم جستجوی ترتیبی

```
Void  seqsearch ( int n, const keytype  S[ ], keytype  x,
                                index &location)
{
     location = ۱;
     while (location <= n && S[location]  != x)
               location++;
     if  (location  >  n )
          location  = ۰ ;
}
```

الگوریتم جستجوی دودویی

Binary search

- شرط انجام این جستجو داشتن یک آرایه مرتب است.
- تقسیم آرایه به دو نیمه و مقایسه با عنصر میانی

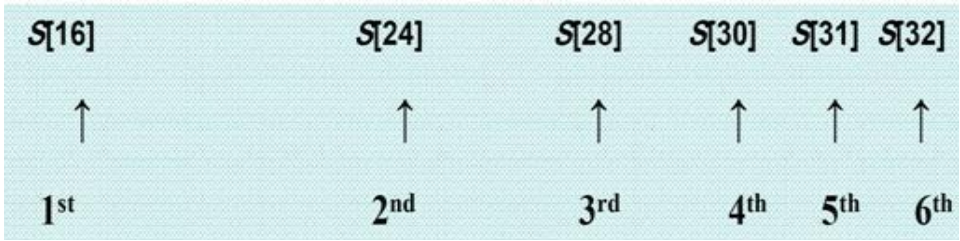
```
void binsearch ( int n,
                 const keytype S[ ],
                 keytype x,
                 index& location )
{
    index low, high, mid;

    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:

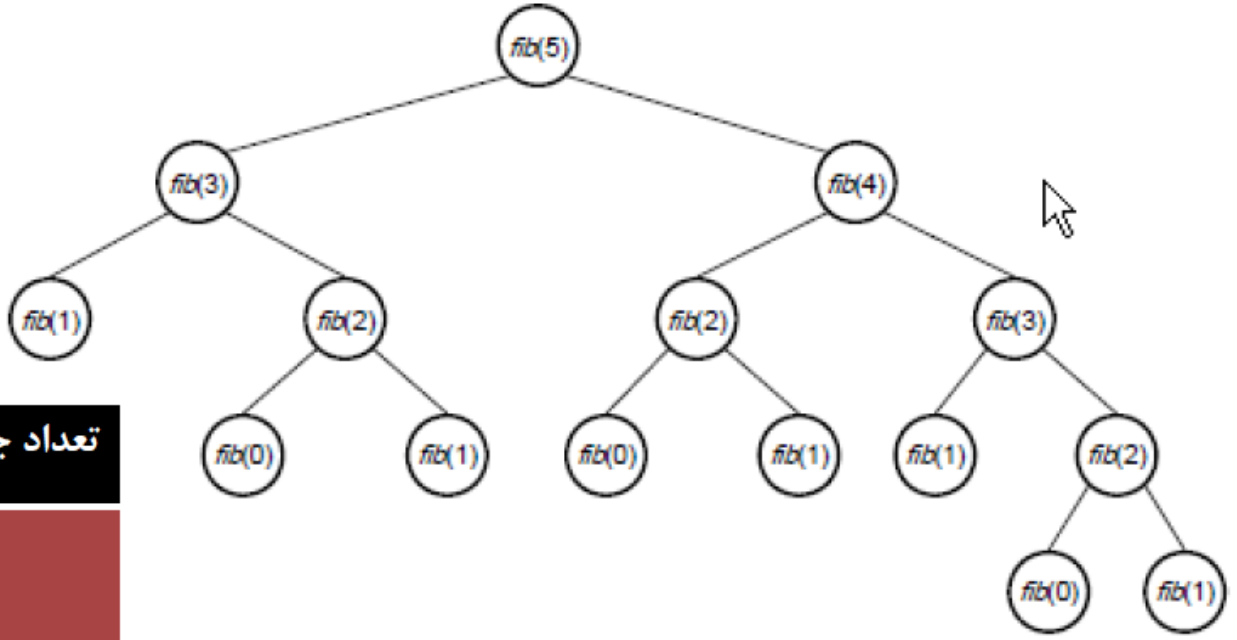


تعداد مقایسه های انجام شده توسط جستجوی ترتیبی (n)	تعداد مقایسه های انجام شده توسط جستجوی دودویی (lg n +۱)	اندازه آرایه
۱۲۸	۸	۱۲۸
۱۰۲۴	۱۱	۱۰۲۴
۱۰۴۸۵۷۶	۲۱	۱۰۴۸۵۷۶
۴۲۹۴۹۶۷۲۹۴	۳۳	۴۲۹۴۹۶۷۲۹۴

الگوریتم جمله n اُم فیبوناچی (بازگشتی)

الگوریتم جمله n اُم فیبوناچی (بازگشتی)

- با رسم درخت بازگشتی واضح است که این الگوریتم بسیار ناکارآمد می باشد.



n	تعداد جملات محاسبه شده
۰	۱
۱	۱
۲	۳
۳	۵
۴	۹
۵	۱۵
۶	۲۵

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

الگوریتم جمله n اُم فیبوناچی (بازگشتی)

- **T(n) : تعداد جملات محاسبه شده در درخت بازگشتی**
- با افزودن ۲ واحد به n، تعداد جملات درخت بیش از ۲ برابر می شود.

$$
\begin{aligned}
T(n) &> 2 * T(n-2) && \text{when } n \geq 2 \\
&> 2 * 2 * T(n-4) \\
&> 2 * 2 * 2 * T(n-6) \\
&\dots \\
&> \underbrace{2 * 2 * \dots * 2}_{n/2 \text{ بار}} * T(0) = 2^{n/2}
\end{aligned}
$$

- $T(n) > ۲^{n/۲}$

محاسبه جمله nام فیبوناچی (تکراری)

- برای تعیین جمله n-اُم دنباله، هر یک از (n-۱) جمله نخست را فقط یک بار محاسبه می کند.
- **الگوریتمی کارآمد**

```
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+۱)	روش بازگشتی ($۲^{n/۲}$)	زمان اجرای تکراری	زمان اجرای بازگشتی
۴۰	۴۱	۱,۰۴۸۵۷۶	۴۱ns	۱۰۴۸ µs
۶۰	۶۱	۱.۱*۱۰^۹	۶۱ns	۱ s
۱۰۰	۱۰۱	۱.۱*۱۰^۱۵	۱۰۱ns	۱۳ days
۱۲۰	۱۲۱	۱.۲*۱۰^۱۸	۱۲۱ns	۳۶ years
۱۶۰	۱۶۱	۱.۲*۱۰^۲۴	۱۶۱ns	۳.۸*۱۰^۷ years
۲۰۰	۲۰۱	۱.۳*۱۰^۳۰	۲۰۱ns	۴*۱۰^۱۳ years

✓ با فرض محاسبه هر جمله در مدت ۱ نانوثانیه

Note: ۱ µs = ۱۰^-۶ s

۱ ns = ۱۰^-۹ s

دیگر الگوریتم های فیبوناتچی

```
Function fib2(n)
        i ← 1
        j ← 0
        for k ← 1 to n do
                j ← i + j
                i ← j - i
        end
        return (j)
end
```

```
function fib3(n)
        i ← 1
        j ← 0
        k ← 0
        h ← 1
        While n > 0 do
                If n is odd
                        Then [ t ← j.h
                               j ← i.h + j.k + t
                               i ← i.k + t]
                t ← h^2
                h ← 2.h.k + t
                k ← k^2 + t
                n ← n div 2
        end
        Return (j)
End
```

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 است.

تحلیل پیچیدگی‌های زمان و فضا

- **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 = ۱ ; i < = n ; i + + )   a + + ;
```

  - O(۱) می‌باشد.
- اگر پیچیدگی فضای یک الگوریتم به n وابسته نباشد، پیچیدگی آن را ثابت فرض کرده و از مرتبه O(۱) می‌دانیم.

پیچیدگی‌های زمان و فضا

- **پیچیدگی زمان (Time Complexity):**
  - محاسبه کارآیی بر حسب زمان
  - مستقل از کامپیوتر، زبان برنامه نویسی و برنامه نویس
  - تحلیل پیچیدگی زمانی یک الگوریتم، تعیین تعداد دفعاتی است که عملیات اصلی الگوریتم (مقایسه، جمع، ضرب و ...) بر حسب اندازه ورودی انجام می شود.

❖ برای تحلیل پیچیدگی یک الگوریتم تابعی به نام $T(n)$ در نظر می گیریم که در آن n اندازه ورودی می باشد.

❖ تحلیل پیچیدگی زمانی الگوریتم ها

- غیر بازگشتی
  - حلقه ها، دستورات کنترلی، حلقه های تودرتو و ...
  - اگر در الگوریتم‌های غیربازگشتی تعداد تکرار دستورات حلقه تکرار به n بستگی نداشته باشد، $T(n)$ (پیچیدگی زمان) مقداری ثابت خواهد بود.
- بازگشتی

تحلیل پیچیدگی زمانی

- اندازه ورودی
  - در بسیاری از الگوریتم ها یافتن میزانی منطقی از اندازه ورودی آسان است.
  - مثال:
    - جستجوی ترتیبی
    - محاسبه مجموع عناصر آرایه
    - مرتب سازی تعویضی
    - جستجوی دودویی
    - ضرب ماتریس ها
  - در برخی از الگوریتم ها بهتر است اندازه ورودی را برحسب دو عدد بسنجیم:
    - مثلا اگر ورودی الگوریتم یک **گراف** باشد:
      - اندازه وروی را برحسب **تعداد رئوس و تعداد یال ها** می سنجیم.

تحلیل پیچیدگی زمانی

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: جستجوی ترتیبی و جستجوی دودویی لیستی به طول n عنصر
  - در هر مرحله 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 انجام می دهند، می توان کارآیی این دو الگوریتم را به ازای هر مقدار از n مقایسه نمود.

محاسبه زمان اجرای الگوریتم

پیدا کردن max و min یک ارایه

```
1) Procedure Smaxmin( A,n,max,min )
2)     Max,min ← A(1)
3)     For i ← 2 to n do
4)         If A(i) > max
5)             Then max ← A(i)
6)         If A(i) < min
7)             Then min ← A(i)
8)     End
9) End
```

1- شماره دستورالعمل	2- تعداد دفعات تکرار	3- زمان مصرفی هر اجرا	2*3
1	1		
2	2		
3	n		
4	n-1		
5	n-1حداکثر		
6	n-1		
7	n-1 حداکثر		
8	n-1		
9	1		
	مجموع = شاخص زمان مصرفی		مجموع = زمان مصرفی

هزینه زمانی تابع فاکتوریل

```
(1)     int    Factorial( int  n )
        {
(2)             int  fact= 1 ;
(3)             for( int  i=2  ; i<= n ; i ++  )
(4)                     fact*= i ;
(5)             return fact ;
        }
```

سطر	هزینه	تعداد
2	C_1	۱
3	C_2	n
4	C_3	n – ۱
5	C_4	۱

$$T(n) = C_1 + C_2 n + C_3 (n-1) + C_4$$

تحلیل پیچیدگی در حالات مختلف

- در برخی موارد مانند الگوریتم **مجموع عناصر آرایه**، عمل اصلی همواره به ازای یک نمونه با اندازه ورودی n به یک میزان انجام می شود.

- در برخی موارد دیگر، تعداد دفعات اجرای عمل اصلی نه تنها به اندازه ورودی بلکه به ترکیب و چیدمان مقادیر ورودی نیز بستگی دارد.
  - مثال: در جستجوی ترتیبی (با اندازه ورودی برابر با n)
    - اگر x در اولین مکان آرایه باشد: تعداد مقایسه ها = ۱
    - اگر x در آرایه نباشد: تعداد مقایسه ها = n

- **T(n)**: تعداد دفعاتی که الگوریتم، عمل اصلی را به ازای یک نمونه با اندازه ورودی n انجام می دهد.

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 می شود.***
- **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,

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

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

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

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

$$۲^{\log n} = n$$

$$۴^{\log n} = n^{۲}$$

خصوصیات توابع نمایی(Exponential)

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

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

For all n and $a \geq 1$, the function a^n is monotonically increasing in n. convenient, we shall assume $0^0 = 1$.

تحلیل پیچیدگی زمانی

- 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 اول: فرض کنیم عنصر مورد نظر قطعا در آرایه وجود دارد. در این صورت احتمال اینکه عنصر در مکان k باش برابر ۱/n می باشد.

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

تحلیل پیچیدگی زمانی

- جستجوی خطی (linear search)
  - حالت متوسط
  - حالت دوم: عنصر مورد نظر ممکن است در آرایه وجود نداشته باشد.
    - احتمال وجود عنصر مورد نظر در آرایه برابر با p
    - احتمال وجود آن در خانه k ام برابر با p/n است.
    - احتمال عدم وجود عنصر در آرایه ۱- p

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

محاسبه دقیق هزینه زمانی مرتب سازی درجی

```
1.  void insertionSort(int n, int a[])
2.  {
3.      for(int j = 2; j <= n; j++)
4.      {
5.          // Insert a[j] into the sorted sequence a[1..j-1].
6.          int key = a[j];
7.          int i = j – 1;
8.          while(i > 0 && a[i] > key)
9.          {
10.             a[i+1] = a[i];
11.             i--;
12.         }
13.         a[i+1] = key;
14.     }
15. }
```

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
-	-
-	-

کل زمان اجرای مرتب‌سازی درجی

$$T(n) = c_1 n + c_3(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

پیچیدگی زمانی مرتب سازی درجی

کل زمان اجرای مرتب‌سازی درجی

$$T(n) = c_1 n + c_3(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

- بهترین حالت(best case)
- زمانی است که آرایه از قبل مرتب باشد.
  - tj=۱

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

- 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

$$T(n) = c_1 n + c_3(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

- 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[۱..j-۱] مقایسه شود.
- بنابراین برای $j=۲,۳,...,n$ داریم $t_j=j$

$$T(n) = c_1 n + c_3(n-1) + c_4(n-1) + c_5(\frac{n(n+1)}{2} - 1) + c_6(\frac{n(n-1)}{2}) + c_7(\frac{n(n-1)}{2}) + c_8(n-1)$$

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

- هزینه زمانی مرتب سازی درجی در بدترین حالت، تابعی درجه دوم به صورت $an^۲+bn+c$ می باشد.

پیچیدگی زمانی مرتب سازی درجی

کل زمان اجرای مرتب‌سازی درجی

$$T(n) = c_1 n + c_3(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

- حالت متوسط(average case)
- می توان فرض کرد a[j] از نصف عناصر a[۱..j-۱] کوچکتر است
- در این صورت هر عنصر a[i] باید با نصف عناصر مرتب شده در زیر آرایه a[۱..j-۱] مقایسه شود.
- بنابراین برای $j=۲,۳,...,n$ داریم $t_j=j/۲$
- هزینه زمانی مرتب سازی درجی در بدترین حالت، تابعی درجه دوم به صورت $an^۲+bn+c$ می باشد.
- حالت متوسط در این الگوریتم همانند بدترین حالت می باشد.

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

- **تمرین**

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

۱-۲- الگوریتمی بنویسید که دومین بزرگترین عنصر یک لیست n عنصری را پیدا کند. در بدترین حالت چند مقایسه لازم است؟

۱-۳-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 ، ۱۰۰n 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 n^2 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:

$$\theta(\log n)\ ,\theta(n)\ ,\ \theta(n\log n)\ ,\theta(n^2)\ ,\theta(n^3)\ ,\theta(2^n),\ldots$$

نمادهای مجانبی(asymptotic notation)

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

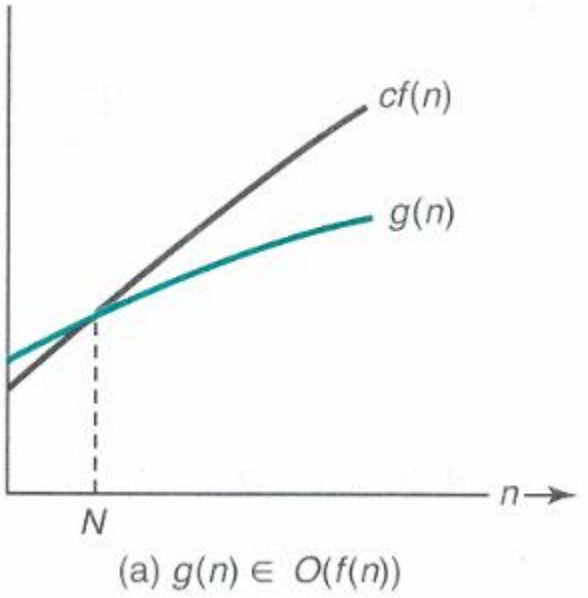
- برای توصیف زمان اجرای الگوریتم از نماد های مجانبی استفاده می کنیم.
- جهت مقایسه الگوریتم‌های مختلف با یکدیگر از نمادهای مجانبی استفاده می کنیم.

نمادهای مجانبی

- Big O (O 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 × *f* (n)

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



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

نمادهای مجانبی

- Big O (O بزرگ)

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

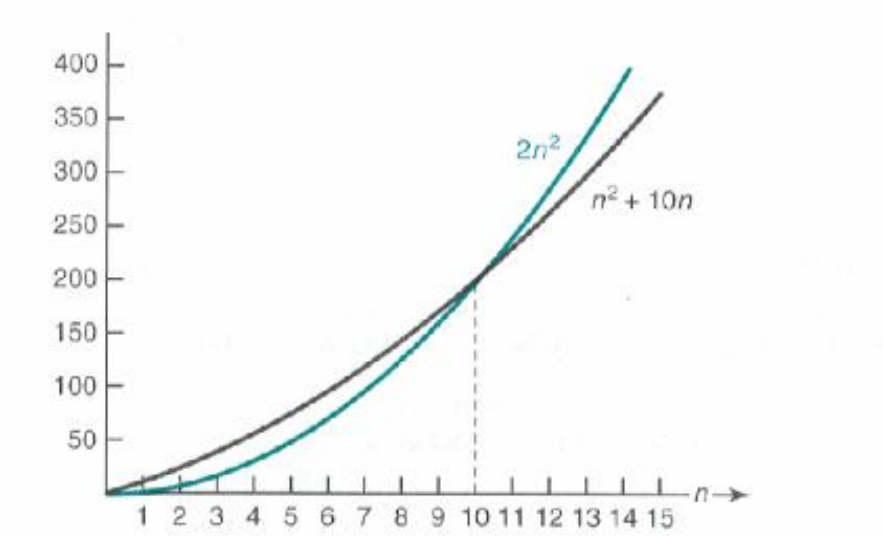


Figure 1.5 • The function $n^2 + 10n$ eventually stays beneath the function $2n$

- **O**ی بزرگ یک حد بالای مجانبی (کران بالا) بر روی یک تابع قرار می دهد.

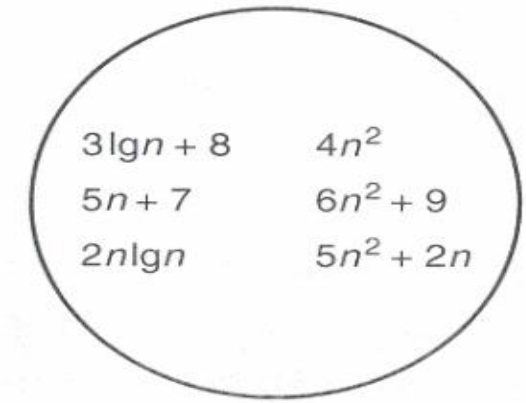
نمادهای مجانبی

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

- $T(n) = n(n\text{-}1)/2 \in O(n^2)$
  $n(n\text{-}1)/2 \leq n(n)/2 = (1/2)n^2$
  $\Rightarrow N = 0, c = 1/2$

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

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



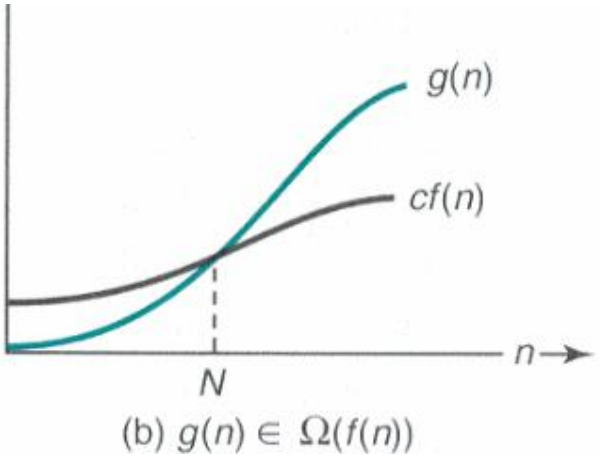
(a) $O(n^2)$

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

نمادهای مجانبی

- 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:

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



(b) $g(n) \in \Omega(f(n))$

- اگر $g(n) \in \Omega(f(n))$ می گوئیم $g(n)$ از امگای $f(n)$ می باشد.
- **Ωیک حد پائین مجانبی(کران پایین حدی) بر روی یک تابع قرار می دهد.**

نمادهای مجانبی

- $5n^2 \in \Omega(n^2)$

  $5n^2 \geq 1 \times n^2 \Rightarrow N = 0,\ c = 1$

- $n^2 + 10n \in \Omega(n^2)$

  $n^2 + 10n \geq n^2 \Rightarrow N = 0,\ c = 1$

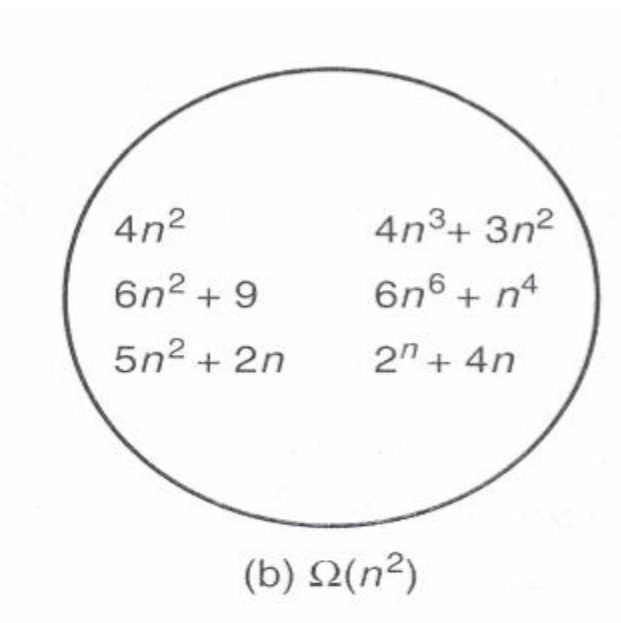
- $T(n) = n(n-1)/2 \in \Omega(n^2)$

  $n \geq 2 \Rightarrow n - 1 \geq n/2 \Rightarrow$

  $n(n-1)/2 \geq (n/2)(n/2) = n^2/4$

  $\Rightarrow N = 2,\ c = 1/4$

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



(b) $\Omega(n^2)$

❖ به طور کلی $\Omega(n^2)$ شامل تمام توابعی است که رشدشان بیشتر یا مساوی n^2 است.

نمادهای مجانبی

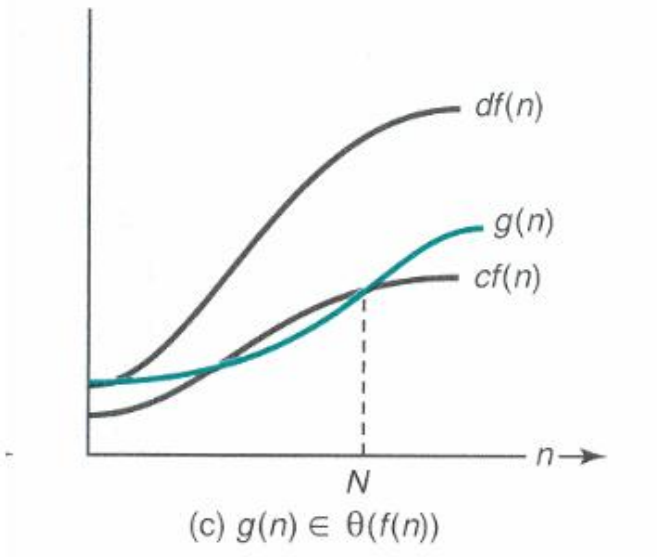
- برای یک تابع پیچیدگی مفروض $f(n)$، داریم:

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

- 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:

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

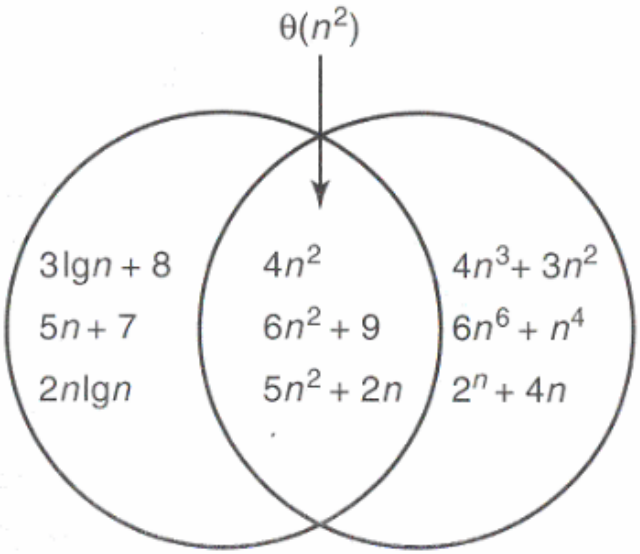
- تابع را از بالا و پایین محدود می کند.



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

نمادهای مجانبی

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



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

نمادهای مجانبی

- o(f(n))
  - اُی کوچک small o-
  - 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: $g(n) < c \times f(n)$
  - ω(f(n))
    - اُمگای کوچک small ω-
  - برای یک تابع پیچیدگی $f(n)$ مفروض، ω(f(n)) عبارت است از مجموعه کلیه توابع پیچیدگی g(n) است که **به ازای هر** ثابت حقیقی مثبت c،ی ک عدد صحیح غیر منفی N وجود دارد طوری که به ازای N <= n داریم:

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

نمادهای مجانبی

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

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

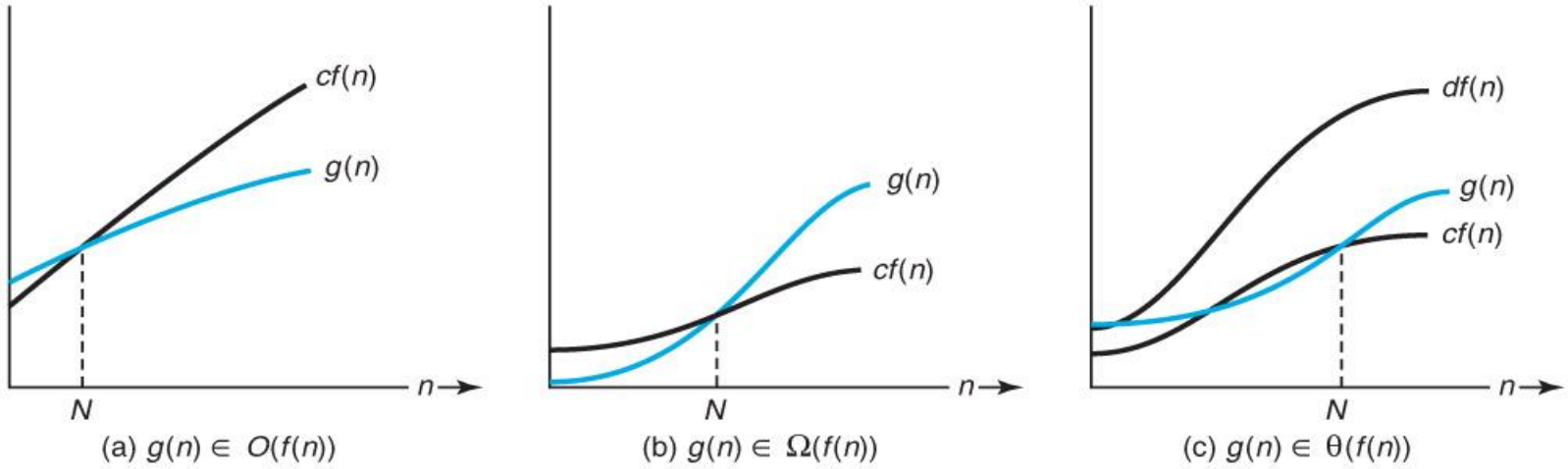
$$g(n) \in \Omega(f(n)) \rightarrow g(n) \geq f(n)$$

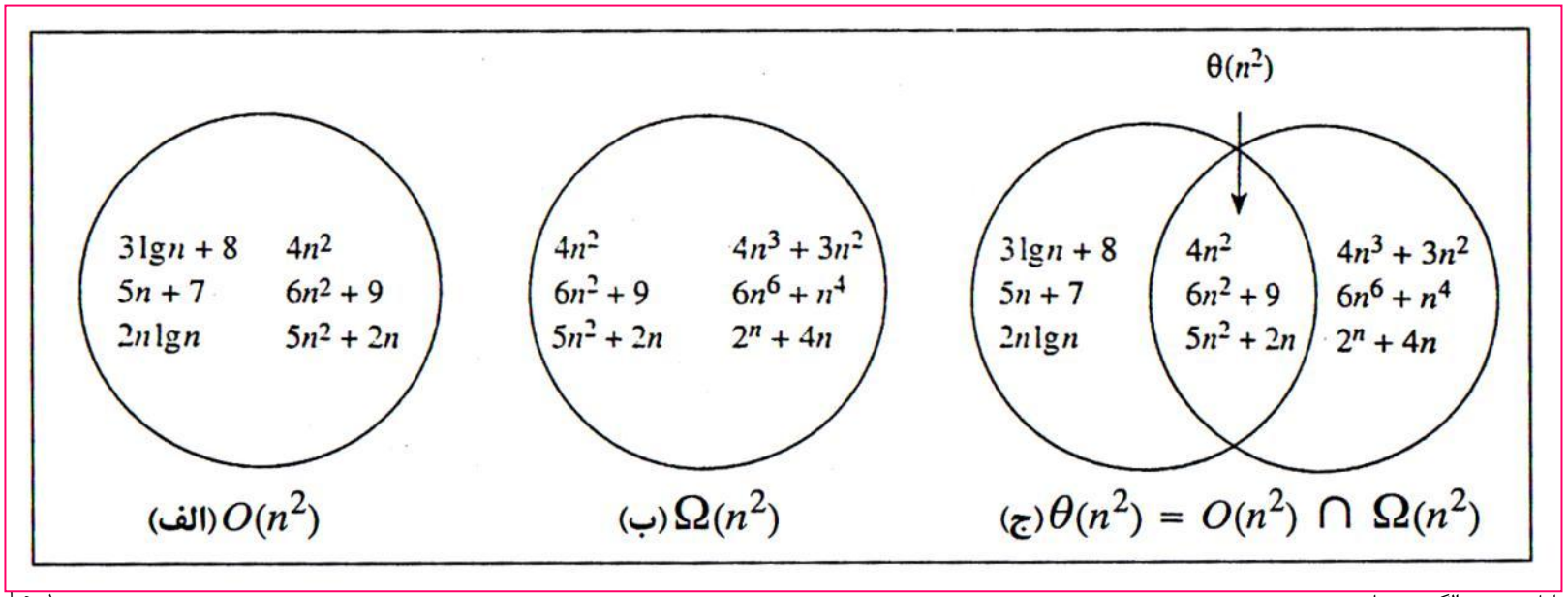
$$g(n) \in \theta(f(n)) \rightarrow g(n) = f(n)$$

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

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

نمادهای مجانبی





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) $6n^2 + 20n \in O(n^3)$

زمان اجرا برای الگوریتم هایی با پیچیدگی زمانی مختلف

n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
۱۰	۰,۰۰۳ μs[*]	۰,۰۱ μs	۰,۰۳۳ μs	۰,۱۰ μs	۱,۰ μs	۱ μs
۲۰	۰,۰۰۴ μs	۰,۰۲ μs	۰,۰۸۶ μs	۰,۴۰ μs	۸,۰ μs	۱ ms[†]
۳۰	۰,۰۰۵ μs	۰,۰۳ μs	۰,۱۴۷ μs	۰,۹۰ μs	۲۷,۰ μs	۱ s
۴۰	۰,۰۰۵ μs	۰,۰۴ μs	۰,۲۱۳ μs	۱,۶۰ μs	۶۴,۰ μs	۱۸,۳ min
۵۰	۰,۰۰۶ μs	۰,۰۵ μs	۰,۲۸۲ μs	۲,۵۰ μs	۱۲۵,۰ μs	۱۳ days
$۱۰^۲$	۰,۰۰۷ μs	۰,۱۰ μs	۰,۶۶۴ μs	۱۰,۰۰ μs	۱,۰ ms	۴×۱۰^{۱۳} years
$۱۰^۳$	۰,۰۱۰ μs	۱,۰۰ μs	۹,۹۶۶ μs	۱,۰۰ ms	۱,۰ s	
$۱۰^۴$	۰,۰۱۳ μs	۱۰,۰۰ μs	۱۳۰,۰۰۰ μs	۱۰۰,۰۰ ms	۱۶,۷ min	
$۱۰^۵$	۰,۰۱۷ μs	۰,۱۰ ms	۱,۶۷۰ ms	۱۰,۰۰ s	۱۱,۶ days	
$۱۰^۶$	۰,۰۲۰ μs	۱,۰۰ ms	۱۹,۹۳۰ ms	۱۶,۷۰ min	۳۱,۷ years	
$۱۰^۷$	۰,۰۲۳ μs	۰,۰۱ s	۲,۶۶۰ s	۱,۱۶ days	۳۱,۷۰۹ years	
$۱۰^۸$	۰,۰۲۷ μs	۰,۱۰ s	۲,۶۶۰ s	۱۱۵,۷۰ days	$۳,۱۷ \times ۱۰^۷$ years	
$۱۰^۹$	۰,۰۳۰ μs	۱,۰۰ s	۲۹,۹۰۰ s	۳۱,۷۰ years		

[*] ۱ ns = $۱۰^{-۹}$ second.

[†] ۱ ms = $۱۰^{-۳}$ second.

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 :

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

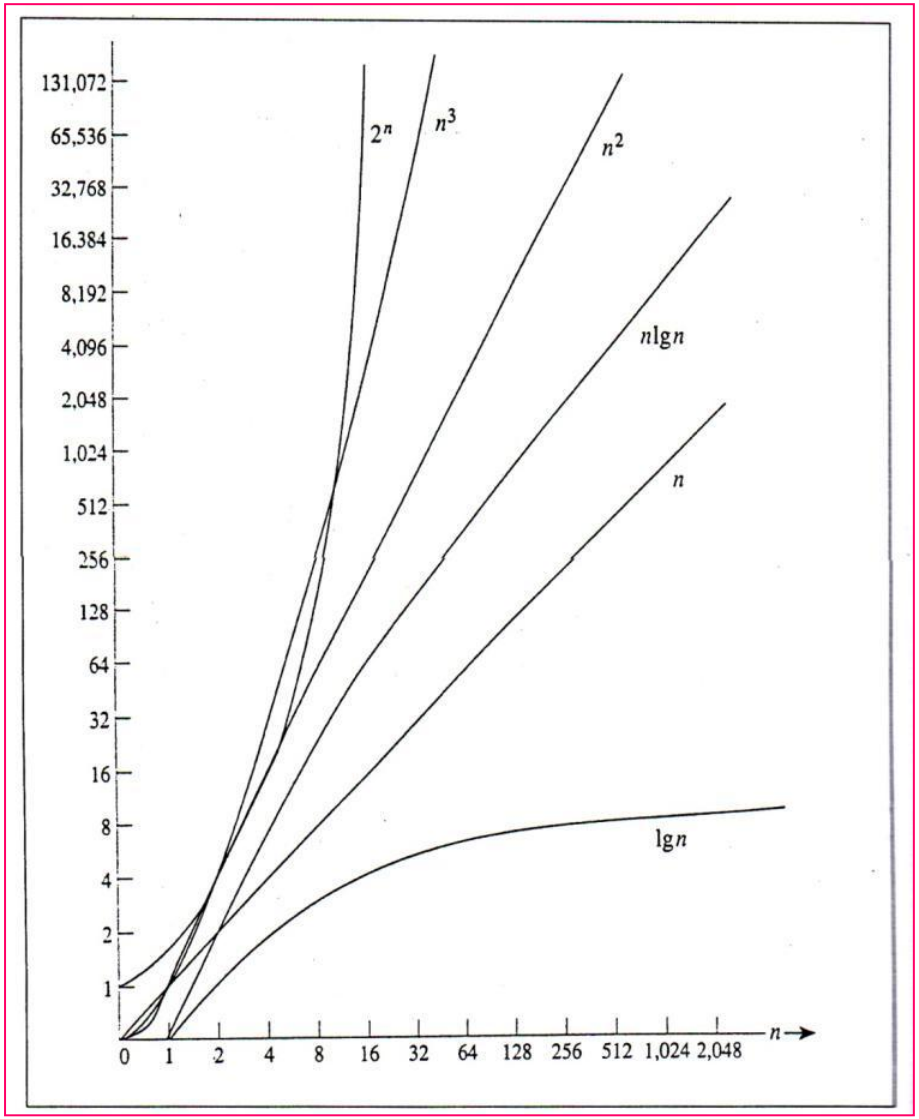
ترتیب رشد توابع

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.

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

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

نمودارهای پیچیدگی



عبارت ریاضی	نسبت سرعت رشد
$g(n)=O(f(n))$	سرعت رشد $f(n)$ ≤ سرعت رشد $g(n)$
$g(n)=\Omega(f(n))$	سرعت رشد $f(n)$ ≥ سرعت رشد $g(n)$
$g(n)=\theta(f(n))$	سرعت رشد $f(n)$ = سرعت رشد $g(n)$

نمادهای مجانبی

- خصوصیات نماد ها

$f(n) \in \Omega(g(n))$ اگر و فقط اگر $g(n) \in O(f(n))$

$f(n) \in \theta(g(n))$ اگر و فقط اگر $g(n) \in \theta(f(n))$

$$if\ f(n) = a_m n^m + \cdots + a_1 n + a_0\ then\ f(n) = \begin{cases} O(n^m) \\ \Omega(n^m) \\ \Theta(n^m) \end{cases}$$

$$f(n) = \Theta(g(n)) \Leftrightarrow \begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \end{cases}$$

چند نکته

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

- الگوریتمی سریعتر است که زمان اجرای بدترین حالت آن رشد کمتری داشته باشد.
  - الگوریتمی مفید است که کران بالای آن پایین است.
- در تعیین مرتبه همواره اجازه حذف جملاتی از مرتبه پائین را داریم.

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

چند نکته

- اگر $0 < a < b$، در آن صورت: $a^n = O(b^n)$
- به ازای همه ی مقادیر $a > 0$ داریم : $a^n = O(n!)$
- اگر $a>1$, $b>1$ آنگاه داریم: $\log_a n \in \theta(\log_b n)$

We can relate the rates of growth of polynomials and exponentials by lowing fact. For all real constants a and b such that $a > 1$,

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0 ,$$

from which we can conclude that

$$n^b = o(a^n) .$$

مثال

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

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

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

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

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

حل : گزینه ۱ و ۲ نادرست است.(چرا؟)

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

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

بنابراین گزینه ۴ صحیح است.

کدام عبارت صحیح است؟ (علوم کامپیوتر – دولتی ۸۲)

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.

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 – دولتی85)**

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 :

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

بنابراین گزینه ۱ صحیح می‌باشد.

نکته : از فرمولهای لگاریتمی زیر استفاده شده است :

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

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

۱-۴-نمادهای مجانبی را از نظر خصوصیات بازتابی(reflexivity)، تقارن(symmetry)، تعدی(transitivity) و ترانهاده تقارنی (transpose symmetry) بررسی کنید.

۱-۵- با استفاده از تعریف نمادها موارد زیر را ثابت کنید؟

الف) $\log(n) = O(n)$ ب) $۶n^۲+۲۰n = \Theta(n^۲)$

ج) $۲^n = \Omega(۵^{\log n})$

تمرین

۱-۶- 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

۱-۷-فرض کنید مرتب سازی درجی در $8n^2$ و مرتب سازی ادغام در $64nlogn$ یک ارایه n عنصری را مرتب می کند. برای چه مقادیری از n مرتب سازی درجی سریعتر است.

۱-۸-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.

۱-۹-یک الگوریتم مرتبه $\Theta(nlogn)$ 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

۱-۱۰-تابع $Log^*(n)$ را تعریف کنید.

۱-۱۱-توابع زیر را از نظر رشد مرتب کنید.

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	n^2	$n!$	$(\lg n)!$
$(\frac{3}{2})^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg^* n$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	1
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$\lg^*(\lg n)$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$	$2^{2^{n+1}}$

تمرین

۱-۱۲-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							
توضیحات							

تمرین

۱-۱۳-

3-2 Relative asymptotic growths

Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^ϵ					
b.	n^k	c^n					
c.	$\sqrt{n}$	$n^{\sin n}$					
d.	2^n	$2^{n/2}$					
e.	$n^{\lg c}$	$c^{\lg n}$					
f.	$\lg(n!)$	$\lg(n^n)$					