مرکز آموزش الکترونیکی دانشگاه علم و صنعت ایران

طراحی و تحلیل الگوریتمها

فصل اول: مقدمه ای بر تحلیل الگوریتمها

(جلسه اول)

اهداف یادگیری

- آشنایی با مفهوم الگوریتم
- آشنایی با الگوریتمهای بازگشتی و غیر بازگشتی
- آشنایی با محاسبه پیچیدگی زمانی الگوریتمها
- آشنایی با نمادهای O,θ,Ω
- یادآوری حل روابط بازگشتی و کاربرد آن در محاسبه پیچیدگی زمانی الگوریتمهای بازگشتی

مقدمه (۱)

- ریشه کلمه الگوریتم از "الخوارزمی"، ریاضیدان قرن دوم هجری گرفته شده است
- معنی لغوی: هر روش مخصوص حل یک دسته مسائل
- معنی در علم کامپیوتر: هر عملی که مراحل مختلف انجام کاری را به زبان دقیق و با جزئیات کافی بیان کند به طوریکه ترتیب مراحل و شرط خاتمه عملیات در آن کاملاً مشخص باشد
- طراحی الگوریتمها: روشهای مختلفی برای طراحی الگوریتمها وجود دارد، که هر فصل این درس به معرفی یکی از آنها می پردازد

مقدمه (۲)

- معتبر سازی (Validation): اثبات درستی یک الگوریتم
- تحلیل الگوریتمها (تحلیل مقدم): منظور تخمینی از زمان اجرای الگوریتم و همچنین میزان حافظه مصرفی آن می باشد
- پیاده سازی: بعد از مرحله معتبر سازی و تحلیل، می توان الگوریتم را با یک زبان برنامه سازی پیاده سازی نمود
- تست برنامه: پس از پیاده سازی الگوریتم، می توان آن را بر روی مجموعه ای از داده های معین اجرا و نتیجه را بررسی نمود

الگوریتمهای بازگشتی

- الگوریتمها به دو دسته بازگشتی و غیر بازگشتی تقسیم می شوند
- الگوریتم بازگشتی مستقیم: هرگاه الگوریتم در تعریف خودش، فراخوانی به خود داشته باشد
- الگوریتم بازگشتی غیر مستقیم: الگوریتم دیگری فراخوانده شود که آن الگوریتم به نوبه خود الگوریتم اولیه را فرا بخواند

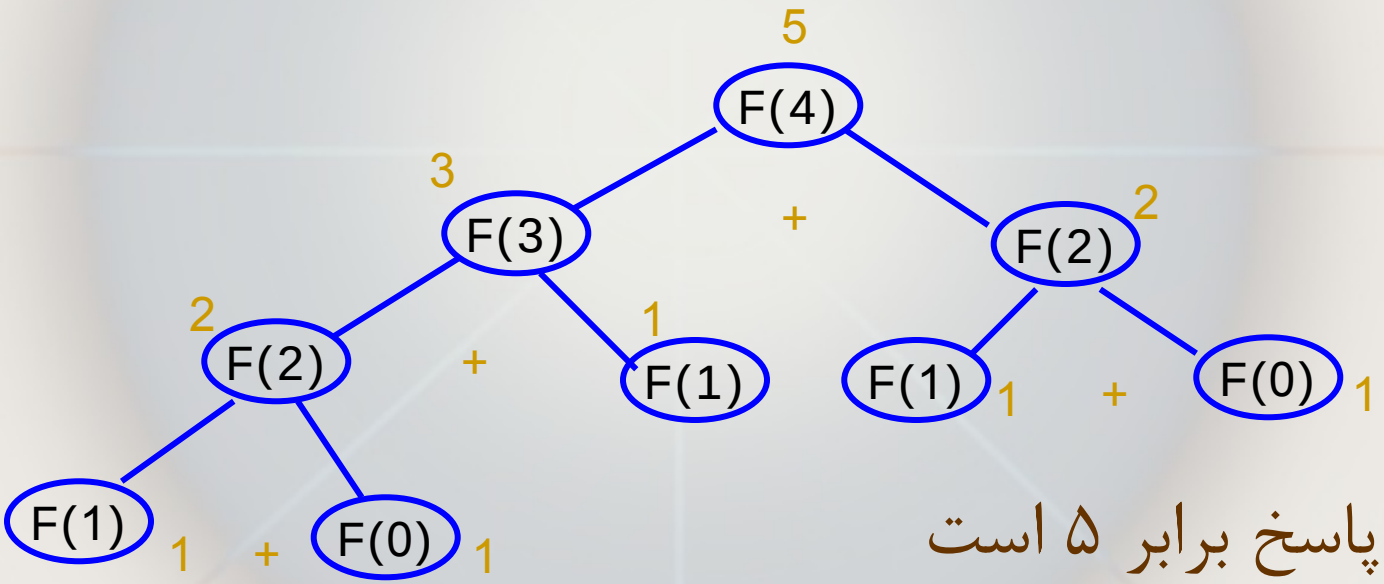
مثال (دنباله فیبوناچی)

- دنباله فیبوناچی: 1,1,2,3,5,...

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

مثال (دنباله فیبوناچی)

- حاصل f(4) چند است؟
- درخت بازگشت را رسم می کنیم



- پاسخ برابر ۵ است

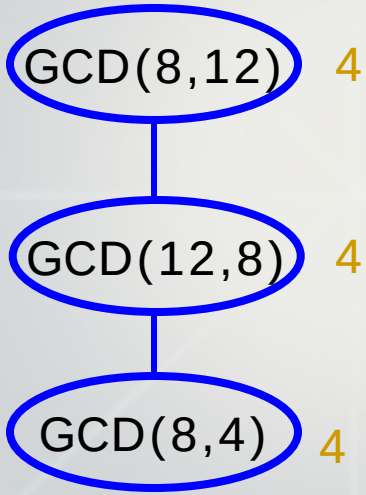
مثال (بزرگترین مقسوم علیه مشترک)

- با فرض اینکه هر دو عدد a وb بزرگتر از صفر هستند

```
int  GCD(int a, int b)
{  if(b==0) return a;
    else return GCD(b, a%b);
}
```

مثال (بزرگترین مقسوم علیه مشترک)

- حاصل GCD(8,12) چند است؟



- پاسخ ۴ می باشد

تحلیل الگوریتمها (پیچیدگی زمانی)

- زمان لازم برای اجرای یک الگوریتم
- زمان کل=زمان اجرای دستورالعمل * تعداد دفعات اجرا
- زمان اجرا بستگی به ماشین، کامپایلر و زبان برنامه نویسی دارد
- در تحلیل الگوریتمها به تعداد دفعات اجرای عمل اصلی بر حسب اندازه مسئله بسنده می کنیم
- منظور از عمل اصلی، بخشی از الگوریتم است که بار محاسباتی در آن قسمت است و در هر الگوریتمی متفاوت است
- منظور از اندازه مسئله، بزرگی مسئله است
- فرض می کنیم زمان اجرای هر دستورالعمل برابر واحد زمان CPU است

درجه بزرگی یک الگوریتم

- درجه بزرگی یک الگوریتم برابر با مجموع تعداد دفعات اجرای عمل اصلی در آن الگوریتم است که به صورت تابعی از اندازه مسئله تعریف می شود
- منظور از تحلیل یک الگوریتم محاسبه درجه بزرگی آن است
- در مثالهای زیر تعداد دفعات اجرای x=x+y (عمل اصلی) چند است؟

```
for(i=1;i<=n;i++)
    x=x+y;
```

درجه بزرگی n

```
i=1 ;  j= 1 to n ;  n بار
i=2 ;  j=2 to n  ;  n-1 بار
i=3 ;  j=3 to n  ;  n-2 بار
...    ;  ...    ;  ...
i=n ;  j=n to n  ;  1 بار
```

$1+2+...+n=n(n+1)/2$ مجموع

```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        x=x+y;
```

درجه بزرگی n^2

```
for(i=1;i<=n;i++)
    for(j=i;j<=n;j++)
        x=x+y;
```

درجه بزرگی n^2

استفاده از نمادها برای درجه بزرگی الگوریتم

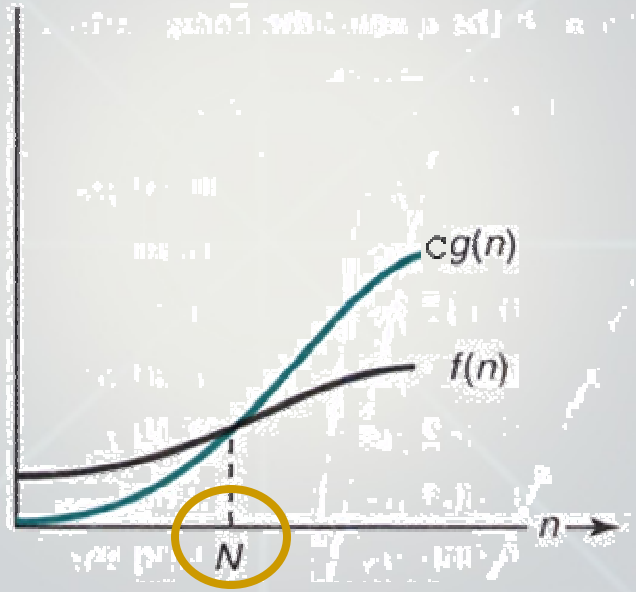
- جهت اندازه گیری و دانستن حدود زمان اجرای الگوریتمها از نمادهای خاص و در عین حال استانداری استفاده می شود که الگوریتمهای مختلف را از نظر درجه بزرگی دسته بندی می کنند
- این نمادها به صورت ریاضی تعریف می شوند و دارای جبر خاصی می باشند
- نمادها عبارتند از:
  - نماد O:
  - نماد θ:
  - نماد Ω:
  - نماد o:
  - نماد W:

نمادهای مجانبی

- نماد O: بنا به تعریف f(n)=O(g(n)) است اگر عددی طبیعی مثل N

  و عدد حقیقی مثل c>0 موجود باشند به گونه ای که به ازای هر n>= N رابطه |f(n)|<=c|g(n)| بر قرار باشد
- با توجه به اینکه f(n) و g(n) زمان اجرای الگوریتمهاست و مثبت هستند، نیازی به استفاده از قدر مطلق نیست
- تابع cg(n) یک کران بالا برای زمان اجرای الگوریتم محسوب می شود

تعبیر هندسی نماد O



مثال

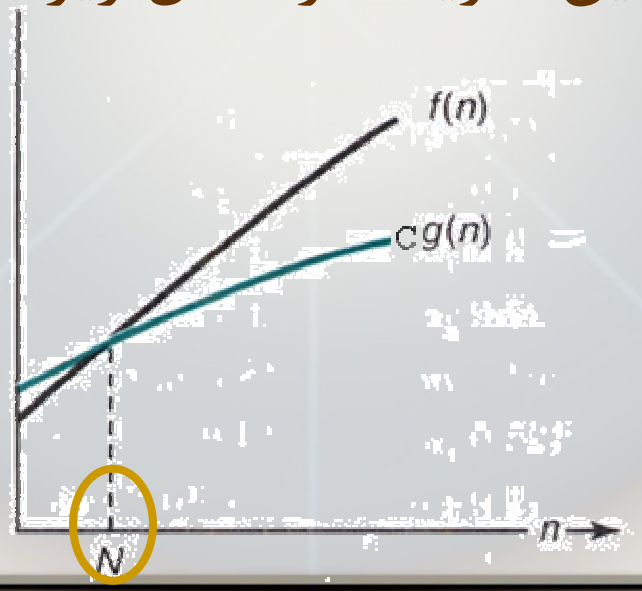
- ثابت کنید $f(n)= 3n^3 + 2n = O(n^3)$
- اگر c=4 انتخاب شود آنگاه باید ثابت کنیم: $3n^3 + 2n <= 4\ n^3$
- چنانچه رابطه فوق را ساده کنیم، باید ثابت کنیم: $2n <= n^3$
- نامساوی فوق همواره به ازای n>=2 بر قرار است. پس N=2 می باشد
- با توجه به اینکه C و N یافته شد، پس مسئله ثابت شده است
- انتخاب C و N بستگی به مسئله دارد
- برای مثال فوق مقادیر متفاوتی برای C و N وجود دارد که می توان توسط آنها مسئله را اثبات نمود. (یافتن فقط یکی از آنها برای اثبات مسئله کافی است)
- در مثال فوق می توان نشان داد که تابع f(n) از $O(n^4)$، $O(n^5)$ و ... می باشد
- برای محاسبه زمان الگوریتمها همیشه از کوچکترین کران بالایی استفاده می شود

چند نتیجه در مورد O

- $A(n)=a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0 \Rightarrow O(n^m)$
- اگر الگوریتمی از k بخش تشکیل شده باشد و زمان بخش kام n_i باشد آنگاه پیچیدگی این الگوریتم برابر بزرگترین زمان است و آنرا به صورت مقابل نشان می دهیم: $n_1 + n_2 + \ldots + n_k$
- اگر T(n)=O(f(n)) و R(n)=O(g(n)) آنگاه R(n).T(n)=O(f(n).g(n))
- اگر تابع f(n) برابر عدد ثابتی باشد، مثلاً f(n)=20 آنگاه f(n)=O(1)

تابع Ω

- بنا به تعریف $f(n)=\Omega(g(n))$ اگر عددی طبیعی مثل N و عددی حقیقی مثل $c>0$ موجود باشند به گونه ای که به ازای هر $n>=N$ رابطه $f(n)>=cg(n)$ برقرار باشد
- تعبیر هندسی این تعریف در شکل زیر نشان داده شده است

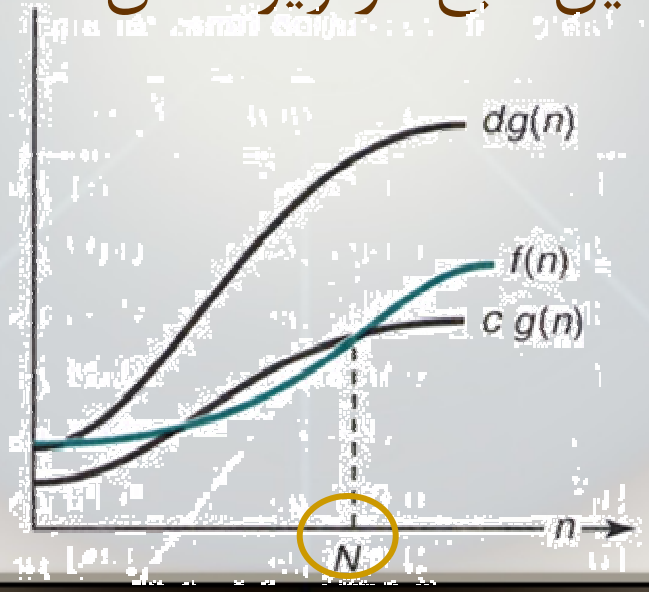


مثال

- ثابت کنید $n^2 + 10n = \Omega(n^2)$
- باید c و N ای پیدا کنیم که به ازای n>=N داشته باشیم:
  $n^2 + 10n <= c\, n^2$
- اگر c=2 انتخاب شود، آنگاه باید ثابت کنیم $10n <= n^2$
- نا مساوی فوق به ازای n>=10 برقرار است. یعنی کافی است N=10 انتخاب شود
- همچنین می توان ثابت کرد که تابع فوق از $\Omega(n)$ و $\Omega(1)$ می باشد
- در تحلیل الگوریتمها، بزرگترین کران پایینی تابع f، به عنوان بهترین زمان اجرای الگوریتم در نظر گرفته می شود

تابع θ

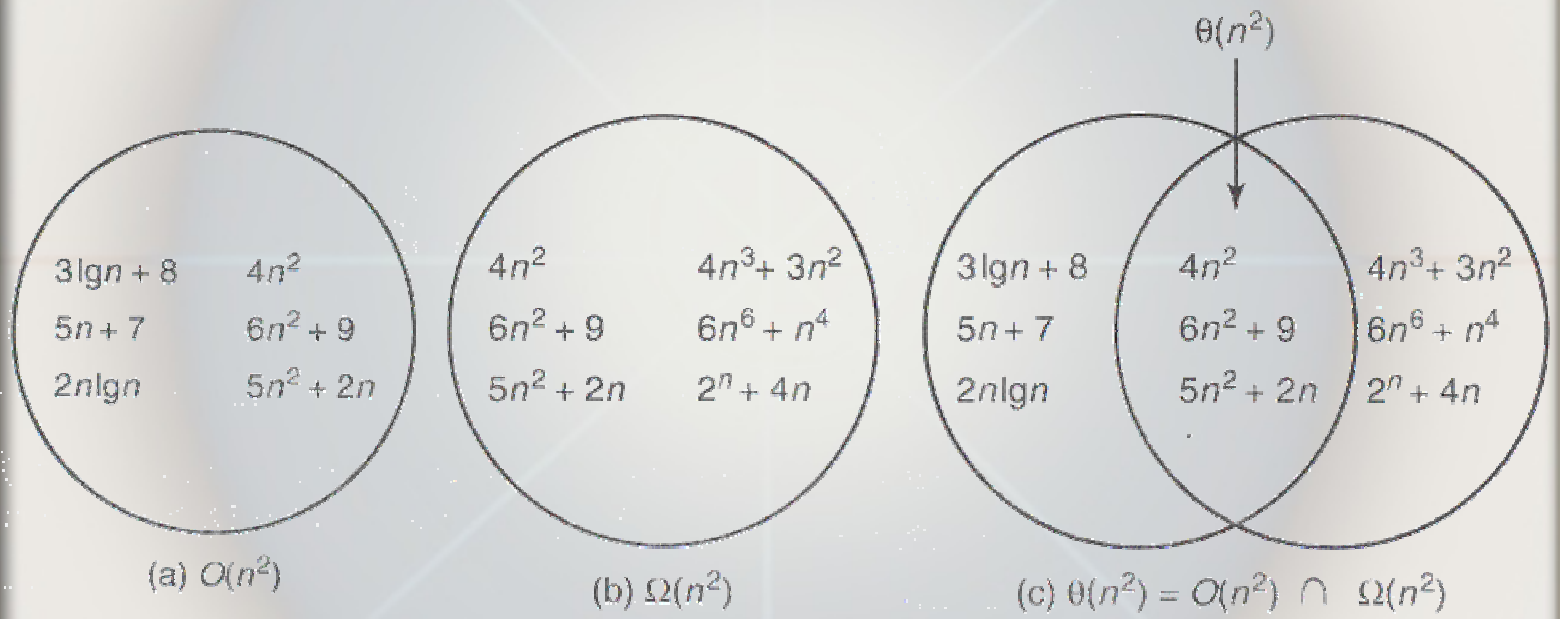
- گوییم $f(n)=\theta(g(n))$ است، اگر عددی طبیعی مثل N و دو عدد حقیقی c و d که بزرگتر از صفرند وجود داشته باشند به طوریکه: $cg(n)<=f(n)<=dg(n)$
- تعبیر هندسی این تابع در زیر نشان داده شده است:



مثال

- ثابت کنید(n^2)$f(n)=n^2 + 10n = \theta$
- یکبار باید ثابت کنیم $f(n)<=d\ n^2$ (در مثال قبل ثابت شد)
- یکبار باید ثابت کنیم $f(n)>=cg(n)$ (مانند دو مثال قبل ثابت می شود)
- در توابع ریاضی داریم: $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- نکته: اگر برای یک الگوریتم میزان O و Ω یکسان بود، آنگاه میزان θ هم برای آن تابع برابر آندو می باشد. البته عکس این مسئله درست نیست

چند مثال



تابع o

* بنا به تعریف f(n)=o(g(n)) هرگاه $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to 0$

* مثال: ثابت کنید: $n^2 + 10n = o(n^3)$

$$\lim_{n\to\infty} \frac{n^2+10n}{n^3} = 0$$

تابع ω

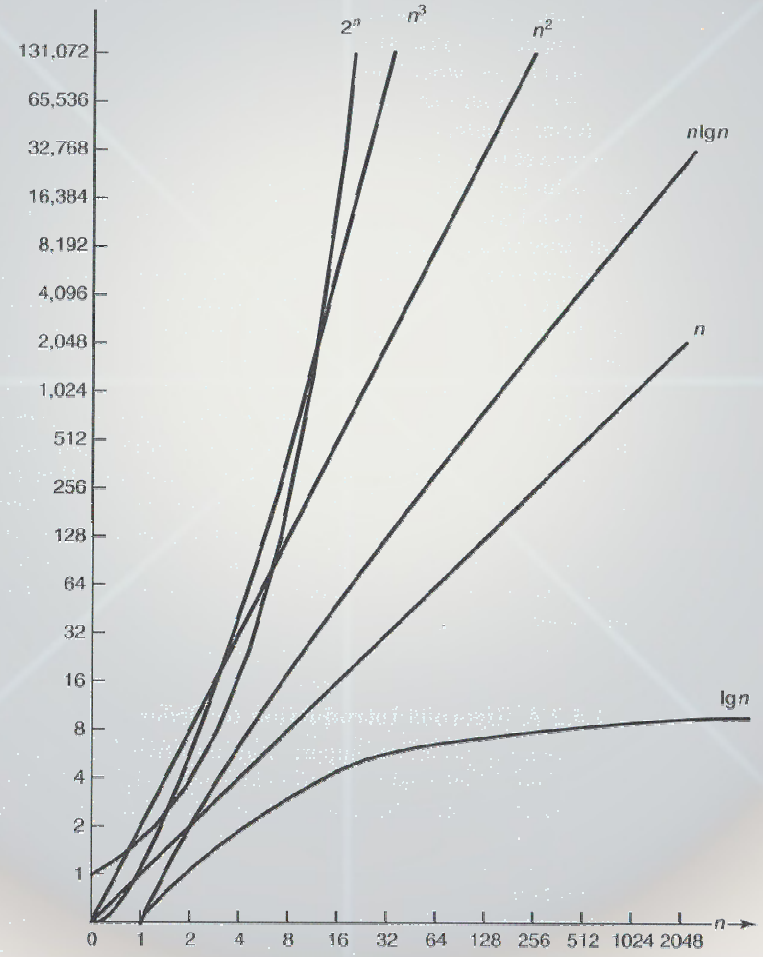
- بنا به تعریف $f(n)=\omega(g(n))$ هرگاه $\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$
- مثال: ثابت کنید $n^2 + 10n = \omega(n)$

$$\lim_{n\to\infty}\frac{n^2+10n}{n}=\infty$$

دسته بندی پیچیدگیها

- استفاده از این نمادها به منظور دسته بندی توابع پیچیدگی می باشد
- به عنوان مثال دو تابع $5n^2 + 100$ و $0.1n^2 + n + 100$ دارای یک Order می باشند و در یک دسته قرار می گیرند
- گروههای متعارف برای توابع پیچیدگی الگوریتمها به صورت زیر می باشد (از کوچک به بزرگ)
  - $\Theta(\lg n)$, $\Theta(n)$, $\Theta(n \lg n)$, $\Theta(n^2)$, $\Theta(n^3)$, $\Theta(2^n)$, $\Theta(n!)$

درجات بزرگی متداول برای الگوریتمها



خواص نمادها

- $f(n)=O(g(n))$ اگر و تنها اگر $g(n)=\Omega(f(n))$
- $f(n)=\theta(g(n))$ اگر و تنها اگر $g(n)=\theta(f(n))$
- اگر $a,b>1$ آنگاه $\log_a n = \Theta(\log_b n)$ (همه توابه لگاریتمی در یک دسته هستند)
- اگر $b>a>0$ آنگاه $a^n = O(b^n)$ (همه توابع نمایی در یک دسته نیستند)
- به ازای هر $a>0$ ، $a^n = o(n!)$ (n! از هر تابع نمایی بزرگتر است) البته $n!=O(n^n)$

تمرینهای حل شده (۱)

- برای تابع زیر میزان O، Ω و θ را تعیین کنید.

$$5n^5 + 4n^4 + 6n^3 + 2n^2 + n + 7$$

- در چند جمله ایها، بزرگترین توان به عنوان O، Ω و θ انتخاب می شود
- پس برای تابع فوق داریم $\theta = O = \Omega = n^5$
- البته با توجه به تعریف این نمادها می توان مقادیر دیگری نیز برای هر یک از نمادها معرفی کرد
- بر اساس تعریف نمادها مقدار فوق قابل اثبات می باشد

تمرینهای حل شده (۲)

- ثابت کنید: $n \lg n \in O(n^2)$
- باید ثابت کنیم: $n \lg n <= c n^2$
- اگر $c=1$ انتخاب شود و طرفین بر n تقسیم شود باید ثابت کنیم: $\lg n <= n$
- نا مساوی فوق به ازای $n>=2$ همواره برقرار است
- درنتیجه $N=2$

تمرینهای حل شده (۳)

- توابع زیر را بر اساس پیچیدگی دسته بندی کنید

$$\log(n!) \quad n\log n \quad n! \quad n^n \quad n^n + \log n$$

- دسته اول: $n\log n$ $\log(n!)$
- دسته دوم: $n!$ n^n $n^n + \log n$
- با توجه به اینکه $n!=1*2*3*\ldots*n$ و $n^n=n*n*\ldots*n$ پس $n!=O(n^n)$
- توجه کنید که $\lg n!=O(\log n^n)=O(n \lg n)$