دانشگاه بین المللی امام خمینی قزوین

دانشکدهٔ فنی و مهندسی

طراحی الگوریتم

(سومین ترم کرونا)

مقدمه

نستوه طاهری جوان

nastooh@aut.ac.ir

معرفی مدرس در ترم و کلاس مجازی!

✓ نستوه طاهری جوان

- لیسانس مهندسی کامپیوتر، نرم افزار
- فوق لیسانس مهندسی کامپیوتر، سخت افزار
- دکترای مهندسی کامپیوتر، شبکه های کامپیوتری
- پسا دکترای مهندسی کامپیوتر، یادگیری ماشین

دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)

- وب سایت شخصی: www.nastooh.com
- صفحه آکادمیک: https://ceit.aut.ac.ir/~nastooh/
- گوگل اسکالر: https://scholar.google.com/citations?user=PmjCrgMAAAAJ
- آدرس ایمیل شخصی: va_nastooh@yahoo.com

بارم بندی

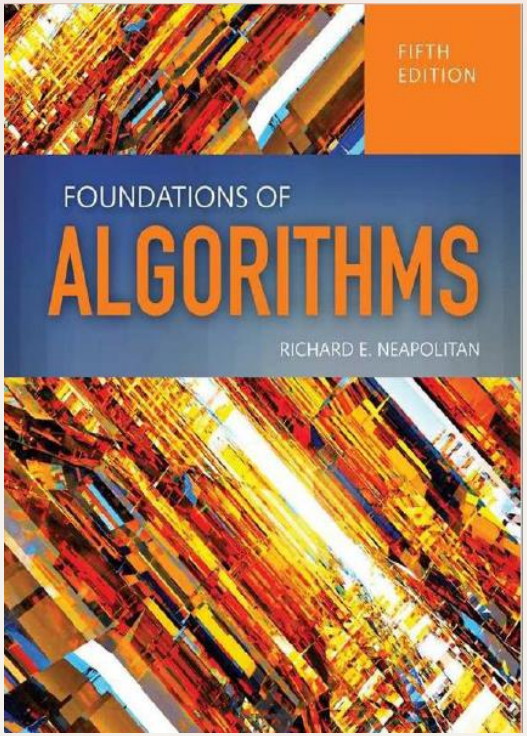
- ✓ امتحان میان ترم
  - ۵ نمره
- ✓ امتحان پایان ترم
  - ۵ نمره
- ✓ تمرین های کلاسی
  - ۵ نمره
- ✓ پروژه پایانی
  - ۵ نمره (بعلاوهٔ نمرهٔ امتیازی)

این بخش بندی ممکن است در طول ترم تغییر کند!

منابع

- ✓ منبع اصلی درس
  - کتاب نیپولیتان:

Richard E. Neapolitan, **“Foundations of Algorithms,”** 5th Ed. (Download Link)



nastooh@aut.ac.ir

مباحث درس در یک نگاه

- ✓ پیچیدگی زمانی و مرتبهٔ اجرایی الگوریتم ها
- ✓ الگوریتم های بازگشتی
- ✓ راهکارهای تقسیم و غلبه
- ✓ راهکارهای برنامه نویسی پویا
- ✓ راهکارهای حریصانه
- ✓ راهکارهای عقبگرد
- ✓ مسائل NP

nastooh@aut.ac.ir

قوانین کلاس

- ✓ تنها قانون کلاس:

مقیاس زمانی کلیهٔ رویدادهای کلاس به ثانیه است!

- منظور رویدادهایی مانند تحویل تمرین، تحویل پروژه، امتحان میان ترم و امثالهم است.

- ○ این به این معنی ست: به هیچ وجه رویدادهای کلاس، تمدید نخواهند شد!!!
  - حتی برای شما دوست عزیز...

به جز این تنها قانون، هیچ محدودیت و قانونی برای کلاس وجود ندارد.

nastooh@aut.ac.ir

راه های تعامل

- ✓ برای انجام امور کلاس از وبسایت کوئرا استفاده خواهد شد.
  - کلیهٔ اطلاع رسانی ها از طریق این سایت انجام خواهد شد.
  - تحویل تمرین ها از طریق این سایت است.
  - آدرس سایت جهت عضویت: https://quera.ir/
    - از نوع دانشجو اکانت بسازید.
  - لینک ورود به کلاس، پس از ساخت اکانت: (برای کلاس طراحی الگوریتم، بهمن ۹۹)
    https://quera.ir/overview/add_to_course/course/7652
    - پسورد مورد نیاز برای ملحق شدن به کلاس، در کلاس آنلاین اعلام خواهد شد.
  - لطفا همین امروز ابتدا در این سایت اکانت ایجاد کنید و سپس با کمک لینک فوق به کلاس درس ملحق شوید.
    - در صورتی که موفق به ایجاد اکانت نشدید، لطفا از طریق ایمیل با مدرس در ارتباط باشید. *nastooh@aut.ac.ir* یا *va_nastooh@yahoo.com*

Time Complexity

- معمولا زمان اجرای یک الگوریتم با افزایش اندازهٔ ورودی (n) افزایش میابد.

- بازدهی الگوریتم ها را با تعیین تعداد دفعاتی که یک **عمل اصلی** انجام می شود، تحلیل می کنیم.
  - انتخاب عمل اصلی، تجربی است.
  - طوری باید انتخاب شود که کار انجام شده توسط الگوریتم تقریبا متناسب با تعداد دفعات اجرای این دستور باشد.

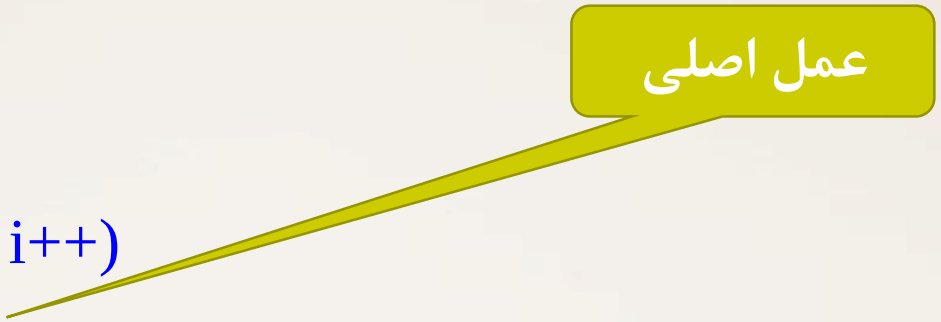
به نظر شما به جز زمان اجرا، چه معیارهای دیگری برای مقایسهٔ الگوریتم ها می توان داشت؟

nastooh@aut.ac.ir

Time Complexity

- پیچیدگی زمانی الگوریتم ها را با T(n) نشان می دهند.
  - نشان می دهد به ازای ورودی n، دستور اصلی چند بار اجرا می شود.

```
int  func (int  n)
{
    int  sum = 0;
    int  i;
    for (i=0; i<n ; i++)
         sum += i;
     return sum;
}
```



nastooh@aut.ac.ir

Time Complexity

- نمونه سوال: تعداد کل مراحل (گام های) برنام زیر را بیابید.

```
int  func (int  n)          0
{                           0
    int  sum = 0;           1
    int  i;                 0
    for (i=0; i<n ; i++)    n+1
         sum += i;          n
     return sum;            1
}                           0

                            2n+3
```


nastooh@aut.ac.ir

Time Complexity

- جمع بندی:
  - گاهی فقط تعداد دفعات تکرار یک دستور خاص را میخوایم.
    - در این صورت $T(n)$ را برای آن دستور حساب می کنیم.
  - گاهی تعداد کل مراحل (گام های) برنامه را می خوایم.
    - در این صورت تعداد کل مراحل همهٔ دستورات را شمارش می کنیم.

nastooh@aut.ac.ir

Time Complexity

- کنکور ارشد، دولتی ۸۳:
  - کدام گزینه تعداد مراحل برنامه زیر را به درستی بیان می کند؟

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

1) $2m + 2n$
2) $mn^2 + m^2n$
3) $m(2n+1)+1$
4) $m(2n+1)-1$

nastooh@aut.ac.ir

Time Complexity

- کنکور ارشد، دولتی ۸۳:
  - کدام گزینه تعداد مراحل برنامه زیر را به درستی بیان می کند؟

```
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++)                m * n
         s[n-1][j]+=s[i][j];              m * (n-1)
    }                                     0
}                                         0
```

$$m+1 + m\,(1+n+n-1) = m + 1 + 2mn = m\,(2n+1)+1$$

nastooh@aut.ac.ir

Time Complexity

- مثال:
  - دستور اصلی در قطعه برنامهٔ زیر چند بار اجرا می شود؟

```
for (j=0; j<n; j++)
    for (i=0; i<j; i++)
        a++;
```


nastooh@aut.ac.ir

Time Complexity

- مثال:
  - دستور اصلی در قطعه برنامهٔ زیر چند بار اجرا می شود؟

```
for (j=0; j<n; j++)
    for (i=0; i<j; i++)
        a++;
```

$$1 + 2 + 3 + 4 + \ldots + n = \frac{n\,(n+1)}{2}$$

nastooh@aut.ac.ir

Time Complexity

- کنکور ارشد، دولتی ۸۰:
  - دستور اصلی در برنامه زیر چند بار اجرا می شود؟

```
for  i = 0 to M do
   for j = 0 to i do
      for k = 1 to M do
         x= x+1;
```

1) $\frac{M^2(M+1)}{2} + 1$

2) $\frac{m^2(m+1)}{2}$

3) $\frac{m(m+1)(m+2)}{2} + 1$

4) $\frac{m(m+1)(m+2)}{2}$

nastooh@aut.ac.ir

Time Complexity

- کنکور ارشد، دولتی ۸۰:
  - دستور اصلی در برنامه زیر چند بار اجرا می شود؟

```
for  i = 0 to M do
   for j = 0 to i do
      for k = 1 to M do
         x= x+1;
```

- حلقهٔ k را مستقل در نظر بگیرید.
- برای دوحلقهٔ بالایی خواهیم داشت:

$$1 + 2 + 3 + \ldots + m + m+1$$

- پس خواهیم داشت:

$$m * \frac{(m+1)(m+2)}{2}$$

nastooh@aut.ac.ir

Time Complexity

- مثال:
  - دستور اصلی در برنامه زیر چند بار اجرا می شود؟

```
i = n;
while (i<1)
   {
       a++;
       i = i / 2;
    }
```

$\lfloor \log_2 n \rfloor$

- با چند n تکرار کنید.

nastooh@aut.ac.ir

Time Complexity

- بهترین حالت، بدترین حالت، متوسط
  - پیچیدگی برای همهٔ توابع همواره ثابت نیست!
  - مثال، جستجوی خطی:

```
for (i= 0; i<n ; i++)
  if (a[i] = = x)
    {
        printf("yes");
        exit();
     }
peintf("No");
```

- ممکن است عنصر مورد جستجو در خانهٔ اول باشد، یا شاید در خانهٔ آخر باشد و یا اصلا موجود نباشد.

nastooh@aut.ac.ir

Time Complexity

- بهترین حالت، بدترین حالت، متوسط
  - برای بدترین حالت از نماد $W(n)$ استفاده می کنیم.
  - برای بهترین حالت از نماد $B(n)$ استفاده می کنیم.
  - و برای حالت میانگین از $A(n)$ استفاده می کنیم.

- در حالت قبلی داریم:

$$W(n) = n$$

$$B(n) = 1$$

$$A(n) = \frac{W(n)+B(n)}{2}$$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- در عمل، محاسبهٔ دقیق تعداد دفعات اجرای دستورات نیاز نیست!
- در مقابل، نیاز به ابزاری داریم تا زمان اجرای الگوریتم ها را به صورت حدودی ولی طبقه بندی شده نشان دهد.
- برای این کار از نماد O استفاده می کنیم.
  - نماد O در عمل نشان می دهد که زمان اجرای یک الگوریتم حدوداً چقدر است!

- **قضیهٔ اصلی مرتبه اجرایی:**

اگر $f(n) = a_m n^m + ... + a_1 n + a_0$ باشد، آنگاه داریم: $f(n) \in O(n^m)$

در واقع می توان گفت به عنوان مثال در تابع $3n^2 + 5n + 6$ سرانجام جملهٔ درجهٔ دوم (یعنی n^2) در این عبارت غالب می شود. یعنی مقادیر جملات دیگر در مقایسه با آن ناچیز خواهند شد.

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- مثال:

- $f(n) = \frac{n}{2}(n+7) \in O(n^2)$
- $f(n) = 5 + 5*9000 \in O(1)$
- $f(n) = \frac{n}{3} + 3n + (n+5) \in O(n)$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- مفهوم رشد توابع:
  - رشد تابع f از رشد تابع g بیشتر است، اگر داشته باشیم:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

➢ در این حالت گوییم: f>g است.

  - رشد دو تابع با هم یکسان است اگر داشته باشیم:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=k$$

➢ در این حالت گوییم: f=g است.

  - و رشد تابع f از g کوچکتر است اگر داشته باشیم:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

➢ در این حالت گوییم: f<g است.

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- مفهوم رشد توابع:
  - مثال: رشد 2^n از n^2 بیشتر است.
  - مثال: رشد $\log_a n$ و $\log_b n$ با هم برابر است.
  - مثال: رشد $\frac{1}{n}$ بیشتر از $\frac{1}{n^2}$ است.

مرتبهٔ اجرایی

- مفهوم رشد توابع:
  - سعی کنید ترتیب زیر را به خاطر بسپارید:

$$\frac{1}{n^2} < \frac{1}{n} < 1 < logn < \sqrt{n} < n < nlogn < n^2 < n^3 <..< 2^n < 3^n <..< n! < nn$$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- O عملا یک کران بالا را برای توابع مشخص می کند.
  - مثال: $5n + 6 \in O(n)$
  - همچنین: $5n + 6 \in O(n^2)$
  - همچنین: $5n + 6 \in O(n!)$
  - البته عموما منظور همان مورد اول است.

می توان گفت

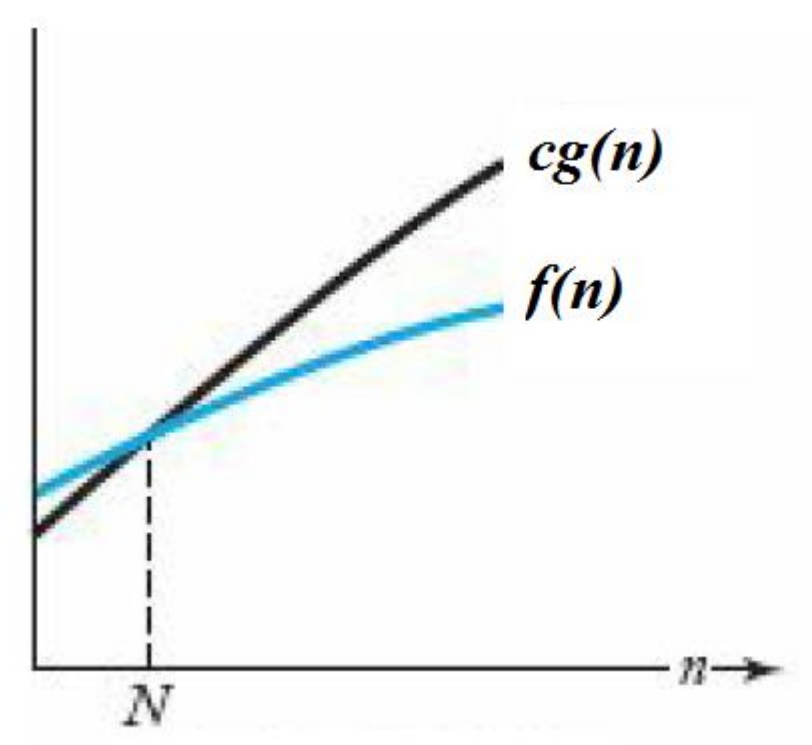
$$f \leq g \quad \approx \quad f(n) \in O(g(n))$$

یعنی اگر رشد تابع g بزرگتر یا مساوی رشد تابع f باشد.

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- O عملا یک کران بالا را برای توابع مشخص می کند.



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

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- چند نکته:
  - توابع با مرتبهٔ نمایی فقط برای ورودی های کوچک عملیاتی هستند و برای ورودی های بزرگ کارآیی ندارند.
  - همچنین توابع با مرتبهٔ فاکتوریل...

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- مثال:
  - الگوریتمی با مرتبهٔ $O(n \log n)$ بر روی کامپیوتری در مدت زمان یک ثانیه اجرا می شود. همان الگوریتم روی کامپیوتر دیگری با سرعت ۱۰۰ برابر، در چه مدتی اجرا می شود؟
    - جواب: یک صدم ثانیه. چرا؟
  - الگوریتمی با اندازهٔ ۱۰ و مرتبهٔ $O(n^2)$ بر روی سیستمی در مدت ۱ ثانیه اجرا می شود، همان مساله با اندازهٔ ۱۰۰ روی همان کامپیوتر در چه مدتی اجرا می شود؟
    - جواب: صد ثانیه. چرا؟

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- تعریف امگای بزرگ Ω
  - برعکس O، می توان گفت یک کران پایین برای تابع مشخص می کند.
  - یعنی عبارات زیر همگی درستند:

$7n^3 + 6n^2 + 3n \in \Omega(n^3)$

$7n^3 + 6n^2 + 3n \in \Omega(n^2)$

$7n^3 + 6n^2 + 3n \in \Omega(n)$

می توان گفت

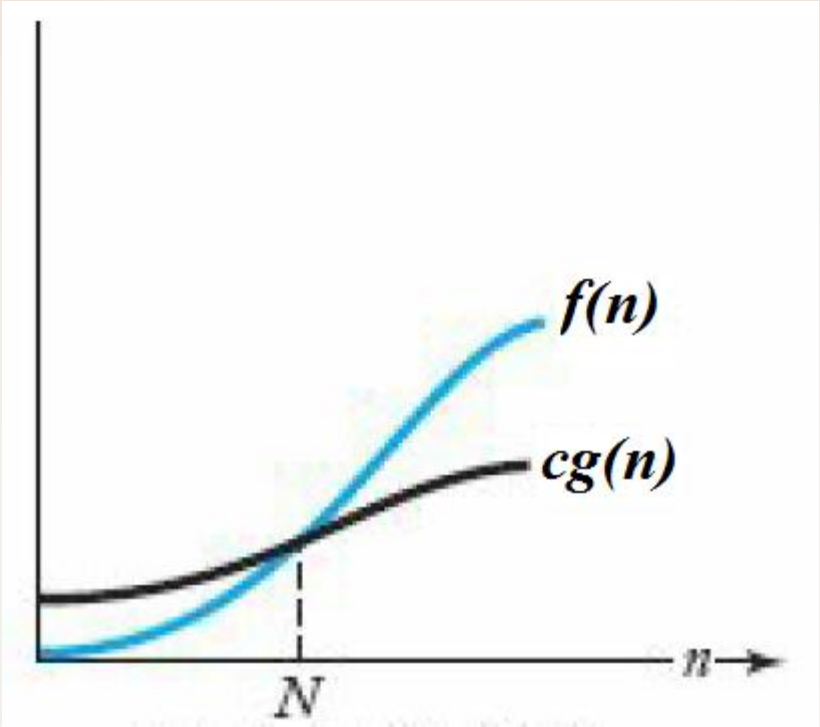
$$f \geq g \approx f(n) \in \Omega(g(n))$$

یعنی اگر رشد تابع g کوچکتر یا مساوی رشد تابع f باشد.

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- تعریف امگای بزرگ Ω



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

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- تعریف تتا θ

می توان گفت

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

یعنی اگر رشد تابع g مساوی با رشد تابع f باشد.

- مثال:

$3n^3 + 4n^2 \in O(n^3)$

$3n^3 + 4n^2 \in O(n^4)$

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

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

$3n^3 + 4n^2 \in \theta(n^3)$

nastooh@aut.ac.ir

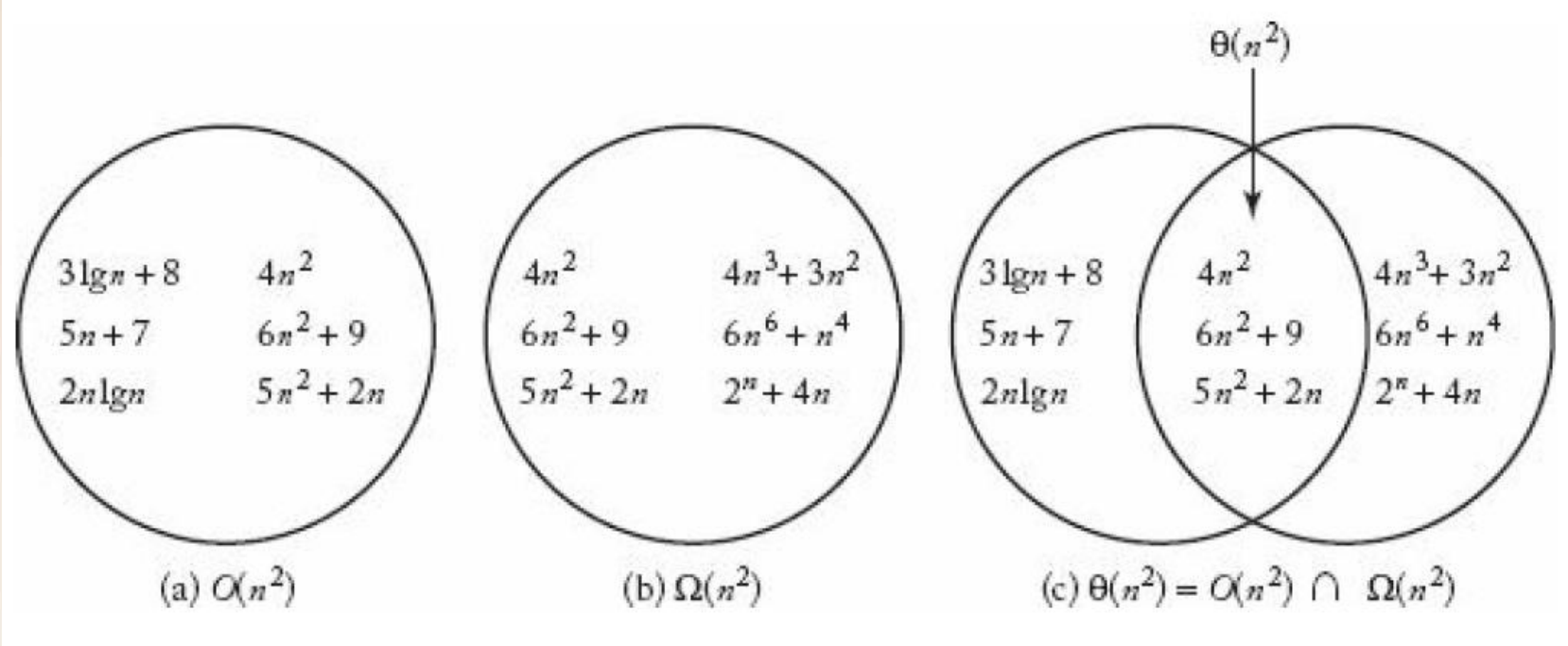
مرتبهٔ اجرایی

- چند نکته:
  - $g(n) \in O(f(n))$ خواهد بود اگر و تنها اگر $f(n) \in \Omega(g(n))$
  - $g(n) \in \Omega(f(n))$ خواهد بود اگر و تنها اگر $f(n) \in O(g(n))$
  - $g(n) \in \theta(f(n))$ خواهد بود اگر و تنها اگر $f(n) \in \theta(g(n))$
  - $g(n) \in \theta(f(n))$ خواهد بود اگر و تنها اگر $f(n) \in \Omega(g(n))$ و $f(n) \in O(g(n))$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- جمع بندی برای O و Ω و θ



شکل از کتاب نیپولیتان

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- نکته تستی:
  - در اغلب تست های کنکور ارشد، منظور از O در واقع همان θ است!
  - در اغلب تست های کنکور ارشد از $=$ بجای $\in$ استفاده می کنند!

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد، دولتی ۷۹:
  - کدام گزینه از نظر افزایش مرتبه از چپ به راست مرتب شده است؟

1. n^{1000}, $n!$, $(1.005)^n$
2. $(1.005)^n$, $n!$, n^{1000}
3. n^{1000}, $(1.005)^n$, $n!$
4. $(1.005)^n$, n^{1000}, $n!$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد، دولتی ۷۹:
  - کدام گزینه از نظر افزایش مرتبه از چپ به راست مرتب شده است؟

1. n^{1000}, n!, $(1.005)^n$
2. $(1.005)^n$, n!, n^{1000}
3. **n^{1000}, $(1.005)^n$, n!**
4. $(1.005)^n$, n^{1000}, n!

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد، آزاد ۸۰:
  - مرتبهٔ بزرگی قطعه کد زیر چیست؟

```
for  i:=1 to  n  do  begin
    for j:=1 to  n  do
        k:=k+1;
    j:=1;
    while j<n  do begin
        k:=k+1;
        j:=j*2;
    end;
end;
```

1. $\theta(nm + n\log n)$
2. $\theta(nm + n^2)$
3. $\theta(nm)$
4. $\theta(n^2)$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد، دولتی ۸۲:
  - کدام عبارت صحیح است؟

1. $(n+1)\,(n^2-2n+1) \in \theta(n)$
2. $(n+1)\,(n^2-2n+1) \in O(2^n)$
3. $(n+1)\,(n^2-2n+1) \in \Omega(n^4)$
4. $(n+1)\,(n^2-2n+1) \in O(n^2 \log n)$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد، دولتی ۸۲:
  - کدام عبارت صحیح است؟

1. $(n+1)\,(n^2-2n+1) \in \theta(n)$
2. **$(n+1)\,(n^2-2n+1) \in O(2^n)$**
3. $(n+1)\,(n^2-2n+1) \in \Omega(n^4)$
4. $(n+1)\,(n^2-2n+1) \in O(n^2 \log n)$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد، دولتی ۸۱:
  - کدام عبارت غلط است؟

1. $\log_2 n \in \theta(\log_{10} n)$
2. $10^n + n^{20} \notin \theta(n^n)$
3. $4n^3 + 5n^2 + 7n \in \Omega(\log_2 n)$
4. $(\log_2 n)! \in \Omega(n!)$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد، دولتی ۸۱:
  - کدام عبارت غلط است؟

1. $\log_2 n \in \theta(\log_{10} n)$
2. $10^n + n^{20} \notin \theta(n^n)$
3. $4n^3 + 5n^2 + 7n \in \Omega(\log_2 n)$
4. $\mathbf{(\log_2 n)! \in \Omega(n!)}$

- نکات:
  - توابع لگاریتمی در هر پایه ای در یک گروه پیچیدگی زمانی هستند.
  - صحیح گزینهٔ ۴: $\mathbf{(\log_2 n)! \in O(n!)}$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- o و ω
  - تعریف o با O کمی تفاوت دارد. داریم:

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

  - در واقع در این تعریف رشد تابع g باید حتما بزرگتر از رشد تابع f باشد. (مساوی ندارد)
  - مثال:

- $2n+1 \in o(n^2)$
- $2n+1 \notin o(n)$
- $4n^2+5n+1 \in o(n^3)$
- $4n^2+5n+1 \in O(n^2)$
- $4n^2+5n+1 \notin o(n^2)$

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- o و ω
  - تعریف ω با Ω کمی تفاوت دارد. داریم:

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

  - آیا می توانید چند مثال برای امگای کوچک بزنید؟

nastooh@aut.ac.ir

مرتبهٔ اجرایی

- کنکور ارشد دولتی، ۸۴:
  - پیچیدگی زمانی اجرای حلقهٔ زیر چیست؟

```
while (n>0)
    n = n / 10;
```

1. O(n)
2. O(n/10)
3. O(1)
4. O(log n)

nastooh@aut.ac.ir

منابع

[1] Richard E. Neapolotan, **“Foundations of Algorithms,”** 5th Ed. (Download Link)

پایان

nastooh@aut.ac.ir