

دانشگاه کاشان

# محاسبه پیچیدگی زمانی

فرشتہ دهقانی  
۹۸-۹۹

## مرواری بر روابط ریاضی

---

$$\lg n = \log_2 n$$

$$\ln n = \log_e n$$

$$\lg^k n = (\lg n)^k$$

$$\lg \lg n = \lg(\lg n)$$

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

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

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



# مرواری بر روابط ریاضی

$$\sum_{i=1}^n i^2 = 1 + 4 + \dots + n^2 = \frac{n * (n+1) * (2n+1)}{6} \approx \frac{n^3}{3}$$

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

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

$$a + (a+d) + (a+2d) + (a+3d) + \dots$$

$$\sum_{k=0}^{n-1} (a + kd) = \frac{n}{2}(2a + (n-1)d)$$

فرض، کنید دنباله هندسی ما بصورت زیر باشد :

$$a + ar + ar^2 + \dots + ar^{(n-1)}$$

هر جمله دنباله به فرم  $ar^k$  که  $k$  مقدار آن از صفر آغاز می شود

در این صورت فرم کلی مجموع جمله ها بصورت زیر است :

$$\sum_{k=0}^{n-1} (ar^k) = a \left( \frac{1 - r^n}{1 - r} \right)$$

a جمله اول تصاعد

r قدر نسبت تصاعد

n تعداد جملات

## مقایسه الگوریتم ها

---

- ❖ وجود الگوریتم های متفاوت برای حل یک مسئله
- ❖ تحلیل الگوریتم ها با هدف های زیر انجام می شود:
- ❖ بررسی و پیش بینی زمان اجرا و میزان حافظه مصرفی یک الگوریتم قبل از پیاده سازی
- ❖ مقایسه الگوریتم های مختلف برای حل یک مسئله از نظر میزان کارایی

# عوامل موثر در سرعت اجرای برنامه ها

---

- ❖ نوع سخت افزار
- ❖ نوع برنامه نویسی (کامپایلر)
- ❖ زبان برنامه نویسی
- ❖ اندازه ورودی
- ❖ ترکیب ورودی (بررسی در سه حالت بهترین، متوسط و بدترین)

# تحليل الگوریتم ها

---

۱- میزان حافظه

۲- زمان اجرای برنامه

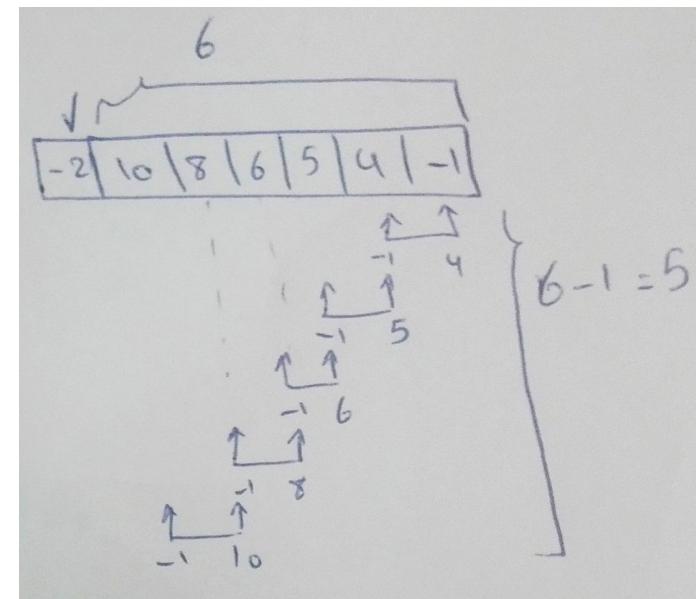
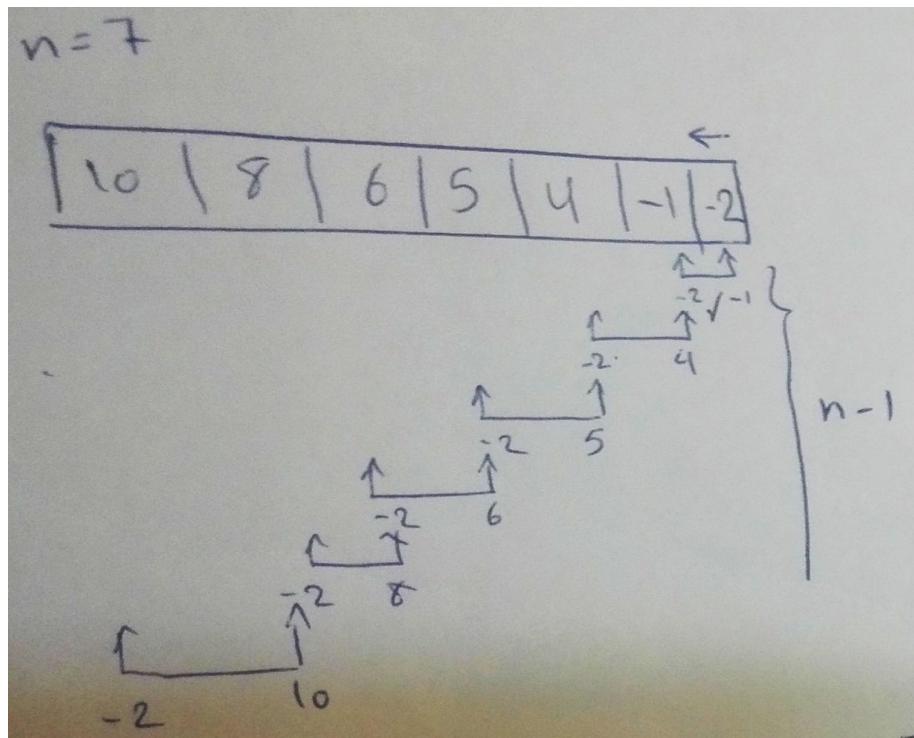
پردازندۀ، کامپایلر، پیچیدگی الگوریتم، اندازه ورودی

## تعداد عملیات

❖ تعداد عملیات‌هایی که هر پردازنده انجام می‌دهد با توجه به ساختار کد و الگوریتم، یکتا و قابل محاسبه (برخلاف زمان اجرای الگوریتم که وابسته به پردازنده‌های مختلف)

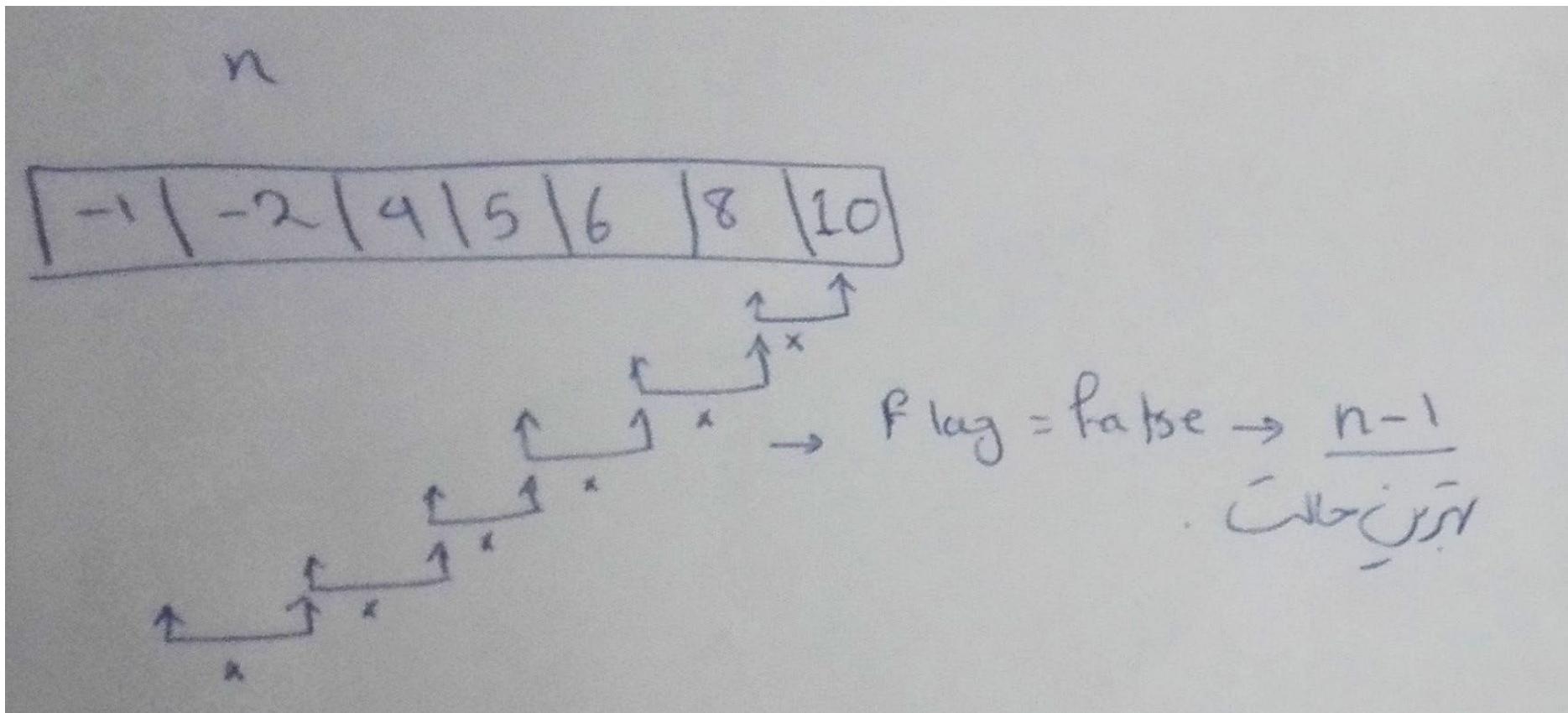
❖ رابطه مستقیم تعداد عملیات‌ها با زمان اجرای الگوریتم  
❖ وابسته به موارد مختلفی مثل تعداد عناصر ورودی و یا بیشینه‌ی اعداد ورودی و ...

# بررسی الگوریتم مرتب سازی حبابی (بدترین حالت)



$$\begin{aligned} &\rightarrow (n-1) + (n-2) + (n-3) + \dots + 1 \\ &= \frac{n(n-1)}{2} \end{aligned}$$

# بررسی الگوریتم مرتب سازی حبابی (بهترین حالت)



# روش های تحلیل الگوریتم از نظر مرتبه زمانی (تعداد عملیات)

---

- ❖ ۱- روش تحلیل تفصیلی
- ❖ ۲- جایگزینی و تکرار
- ❖ ۳- قضیه اصلی
- ❖ ۴- تغییر متغیر
- ❖ ۵- حدس و استقرای ریاضی
- ❖ ۶- درخت بازگشت

## ۱- روش تحلیل تفصیلی

---

مناسب برای برنامه های غیربازگشتی

- ❖ ۱- محاسبه تعداد تکرار دستورات برنامه بر حسب اندازه ورودی و به ازای هر دستور
- ❖ ۲- فرض کردن یک زمان ثابت برای اجرای یک دستور
- ❖ ۳- ضرب موارد یک و دو به ازای هر دستور و جمع حاصل آن ها

# مثال - عملگرها

---

- ❖ Each operation in an algorithm (or a program) has a cost.  
→ Each operation takes a certain amount of time.

count = count + 1; → take a certain amount of time, but it is constant

**A sequence of operations:**

count = count + 1;	Cost: $c_1$
sum = sum + count;	Cost: $c_2$

→ **Total Cost =  $c_1 + c_2$**

## مثال - دستورات شرطی

---

	<u>Cost</u>	<u>Times</u>
if (n < 0)	c1	1
absval = -n	c2	1
else		
absval = n;	c3	1

Total Cost  $\leq c1 + \max(c2, c3)$

## مثال - حلقة

	<u>Cost</u>	<u>Times</u>
i = 1;	c1	1
sum = 0;	c2	1
while (i <= n) {	c3	n+1
i = i + 1;	c4	n
sum = sum + i;	c5	n
}		

$$\text{Total Cost} = c1 + c2 + (n+1)*c3 + n*c4 + n*c5$$

→ The time required for this algorithm is proportional to n

# مثال - حلقه تو در تو

	<u>Cost</u>	<u>Times</u>
i=1	c1	1
sum = 0	c2	1
while i <= n	c3	$n+1 (\sum_{i=1}^n 1 + 1 = n + 1)$
j=1	c4	n
while j <= n	c5	$n * (n+1) (\sum_{i=1}^n \sum_{j=1}^n 1 + \sum_{i=1}^n 1 = n^2 + n)$
sum = sum + i	c6	$n * n$
j = j + 1	c7	$n * n$
i = i + 1	c8	n

**Total Cost = c1 + c2 + (n+1)\*c3 + n\*c4 + n\*(n+1)\*c5+n\*n\*c6+n\*n\*c7+n\*c8**

→ The time required for this algorithm is proportional to  $n^2$

# مثال - حلقه تو در تو

	<u>Cost</u>	<u>Times</u>	<u>Time</u>
for ( <u>i=1; i&lt;=n; i++</u> )	c1	n+1	<u>for (i=1; i&lt;=n; i++)</u>
for ( <u>j=1; j&lt;=i; j++</u> )	c2	$\sum_{j=1}^n (j+1)$	<u>for (j=1; j&lt;=i; j++)</u>
for ( <u>k=1; k&lt;=j; k++</u> )	c3	$\sum_{j=1}^n \sum_{k=1}^j (k+1)$	<u>for (k=1; k&lt;=j; k++)</u>
x=x+1;	c4	$\sum_{j=1}^n \sum_{k=1}^j k$	<u>x = x+1</u>
T(n)	$= c1*(n+1) + c2 * \left( \sum_{j=1}^n (j+1) \right) + c3 * \left( \sum_{j=1}^n \sum_{k=1}^j (k+1) \right) + c4 * \left( \sum_{j=1}^n \sum_{k=1}^j k \right)$ $= a*n^3 + b*n^2 + c*n + d$		
→ So, the growth-rate function for this algorithm is	$\mathbf{O(n^3)}$		

## تخمین یک مجموعه گسته

- س. چگونه می‌توان یک مجموع گسته را تخمین زد؟
- ج. مجموع را با یک انتگرال جایگزین کنید.

$$1 + 2 + \dots + N.$$

$$\sum_{i=1}^N i \cong \int_{x=1}^N x dx \sim \frac{1}{2} N^2$$

$$1 + 1/2 + 1/3 + \dots + 1/N.$$

$$\sum_{i=1}^N \frac{1}{i} \cong \int_{x=1}^N \frac{1}{x} dx = \ln N$$

$$3 - sum.$$

$$\sum_{i=1}^N \sum_{j=i}^N \sum_{k=j}^N 1 \cong \int_{x=1}^N \int_{y=x}^N \int_{z=y}^N dz dy dx \sim \frac{1}{6} N^3$$

# مثال - نرخ رشد توابع

for ( $i=1; i \leq n; i++$ )  $\rightarrow$  مرتبه  $n+1$

{

$j = n;$

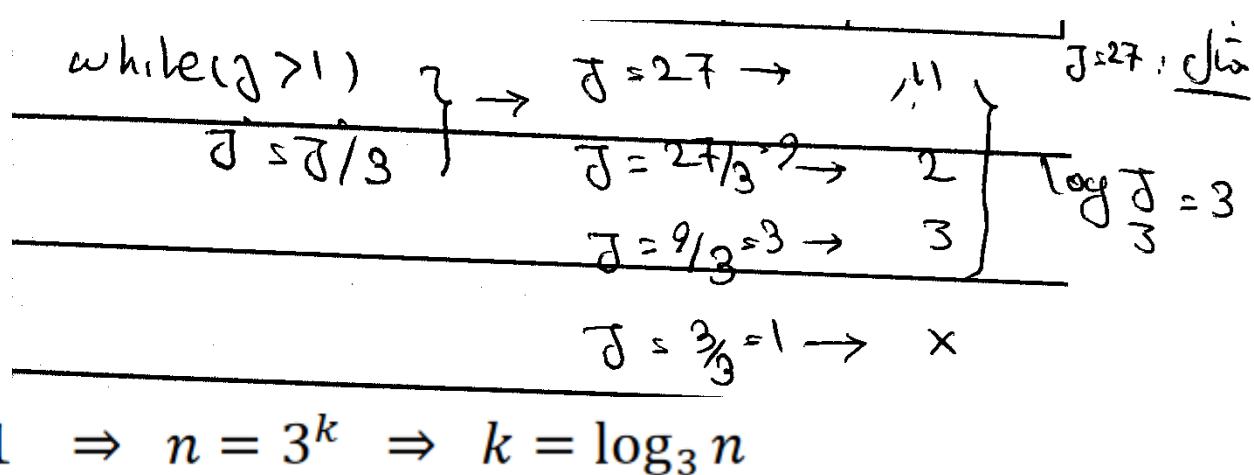
مرتبه  $n$

while ( $j > 1$ )

$j = j/3;$

}

$$\rightarrow \frac{n}{3^k} = 1$$



برای سادگی فرض میکنیم که  $N = 3^k$  است،  
در نتیجه این حلقه  $k$  بار اجرا خواهد شد

$$T(n) = c_1(n+1) + c_2n + c_3n(\log_3^n + 1) + c_4n\log_3^n$$

# مثال

---

*For( $i=1$  ;  $i \leq n$  ;  $i++$ )*

*For( $J=1$  ;  $J \leq i$  ;  $J*=2$ )* ————— *Logn*

*Write(\*)*

$$\sum_{i=1}^n \log 1 + \log + \log 3 + \dots + \log n = \log(1 \times 2 \times 3 \times \dots \times n) = \log n!$$

$\log n! \in \delta(n \log n)$

# بررسی الگوریتم مرتب سازی حبابی

حبل مرتب سازی حبابی (حلقه)

$$\begin{aligned}
 & \text{for } i \leftarrow 1 : N \\
 & \quad \text{for } j \leftarrow N : (i+1) \\
 & \quad \quad \vdots \\
 & \rightarrow \sum_{i=1}^N 1 + 1 = N + 1 \\
 & \rightarrow \sum_{i=1}^N \underbrace{\sum_{j=N}^{i+1} 1}_{(N-i-1+1)} + N = \sum_{i=1}^N (N-i) + N \\
 & = N \sum_{i=1}^N 1 - \sum_{i=1}^N i + N = \\
 & \textcircled{2} \quad N^2 - \frac{N(N+1)}{2} + N = \\
 & \quad \quad \frac{N(N+1)}{2}
 \end{aligned}$$

# بررسی الگوریتم مرتب سازی حبابی

کل مرتبت سازی حبابی ( $O(N^2)$ )

for  $i = 1 : N$

    for  $j = N : (i+1)$

        if ( $A[j] < A[j-1]$ )  $\rightarrow \sum_{i=1}^N (N-i) = \frac{N(N-1)}{2}$

        swap

$\rightarrow N+1$

$\rightarrow \frac{N(N+1)}{2}$

$\rightarrow \frac{N(N-1)}{2}$

$$\Rightarrow c_1(N+1) + c_2\left(\frac{N(N+1)}{2}\right) + c_3\left(\frac{N(N-1)}{2}\right) + c_4\left(\frac{N(N-1)}{2}\right) \approx \frac{N^2}{2}$$

$$c'_1 N^2 + c'_2 N + c'_3 \approx N^2$$

# بررسی الگوریتم مرتب سازی حبابی

*Alg.:* BUBBLESORT( $A$ )

**for**  $i \leftarrow 1$  **to**  $\text{length}[A]$      $c_1$

**do for**  $j \leftarrow \text{length}[A]$  **downto**  $i + 1$      $c_2$

Comparisons:  $\approx n^2/2$     **do if**  $A[j] < A[j - 1]$      $c_3$

Exchanges:  $\approx n^2/2$     **then exchange**  $A[j] \leftrightarrow A[j-1]$      $c_4$

$$\begin{aligned} T(n) &= c_1(n+1) + c_2 \sum_{i=1}^n (n-i+1) + c_3 \sum_{i=1}^n (n-i) + c_4 \sum_{i=1}^n (n-i) \\ &= \Theta(n) + (c_2 + c_3 + c_4) \sum_{i=1}^n (n-i) \end{aligned}$$

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

$$\text{Thus, } T(n) = \Theta(n^2)$$

# تحلیل مرتب سازی درجی

$$\begin{aligned}
 & \text{for } j \leftarrow 2 \text{ to } N \quad \rightarrow \quad \sum_{j=2}^N 1 + 1 = (N-1) + 1 = N \\
 & \text{key} = A[j] \quad \rightarrow \quad \sum_{j=2}^N 1 = N-1 \\
 & i = j-1 \quad \text{and } A[i] > \text{key} \rightarrow \sum_{j=2}^N \left( \sum_{i=2}^{j-1} 1 + 1 \right) \star \\
 & \text{while } i > 0 \quad \rightarrow \quad \sum_{j=2}^N t_j \quad \star \quad t_j
 \end{aligned}$$

- جستجوی ایجاد - صورت حالتی که نمی‌توان صورت حالتی را تغییر داد اما هنوز مسیر را نمی‌توان تغییر داد  
 $\Leftrightarrow t_j = 0$  برای  $i \leq j$   $t_j = 0$  برای  $i > j$

$$\star = \sum_{j=2}^N (0 + 1) = \sum_{j=2}^N 1 = N-1$$

۲- دراین حالت آزادی پرداز زیولی مرتب نمی‌شود، مانند حالت دیگر اینکه  $i \geq j$  باشد

$$*= \sum_{j=2}^N \left( \sum_{i=1}^{j-1} 1 + 1 \right) = \sum_{j=2}^N \underbrace{\sum_{i=1}^{j-1} 1}_{(j-1)} + \sum_{j=2}^N 1 =$$

$$\sum_{j=2}^N (j-1) + N-1 = \underbrace{\sum_{j=2}^N j}_{\frac{N(N+1)}{2}} - \underbrace{\sum_{j=2}^N 1}_{N-1} + (N-1) =$$

$$\frac{N(N+1)}{2} - 1$$

و در دستور داخل حلقه  $\leftarrow (\text{طبق جایگزینی})$

$$* \Rightarrow \sum_{j=2}^N \sum_{i=1}^{j-1} 1 = \frac{N(N+1)}{2} - N + 1 = \frac{N(N-1)}{2}$$

۳- حالت ساده: حلقه while آزادی را بسیار سفید حکم کن و در تابع  $f$  روند گذشت خواهد

$$* = \sum_{j=2}^N \left( \sum_{i=\frac{j-1}{2}}^{j-1} 1 + 1 \right) = \frac{1}{2} \left( \sum_{j=2}^N j + \sum_{j=2}^N 1 \right) + \sum_{j=2}^N 1 \leftarrow \left( \frac{j-1}{2} \right) + 1 = \frac{j+1}{2}$$

$$= \frac{N(N+1)}{4} + \frac{N}{2} - 1 + N - 1 \simeq N^2$$

# مثال-مرتب سازی درجی

---

<code>INSERTION-SORT(<math>A</math>)</code>	<i>cost</i>	<i>times</i>
1 <b>for</b> $j = 2$ <b>to</b> $A.length$	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3       // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 <b>while</b> $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

## مرتب سازی درجی - ادامه

*worst case:* reverse-ordered elements in array.

$$\begin{aligned}\sum_{i=1}^{N-1} i &= 1 + 2 + 3 + \dots + (N-1) = \frac{(N-1)N}{2} \\ &= O(N^2)\end{aligned}$$

*best case:* array is in sorted ascending order.

$$\sum_{i=1}^{N-1} 1 = N - 1 = O(N)$$

*average case:* each element is about halfway in order.

$$\begin{aligned}\sum_{i=1}^{N-1} \frac{i}{2} &= \frac{1}{2} (1 + 2 + 3 \dots + (N-1)) = \frac{(N-1)N}{4} \\ &= O(N^2)\end{aligned}$$

# حالات میانگین، بهترین و بدترین

برای بررسی زمان اجرا معمولاً تعداد عناصر ورودی به عنوان متغیر در نظر گرفته و سپس بررسی الگوریتم در یکی از سه حالت زیر:

❖ **بهترین حالت (Best Case):** اگر اندازه‌ی ورودی الگوریتم را  $n$  در نظر بگیریم، بهترین حالت وقتی است که یک ورودی به اندازه‌ی  $n$  از همه‌ی ورودی‌های همان اندازه‌ی خود زمان اجرای کمتری داشته باشد.

❖ **بدترین حالت (Worst Case):** بر عکس حالت قبل، نوعی از ورودی به اندازه‌ی  $n$  که از همه‌ی ورودی‌های همان اندازه‌ی خود زمان اجرای بیشتری نیاز داشته باشد. (با احتمال بالا، مرتبه اجرای بیشتر الگوریتم‌ها)

❖ **حالت میانگین (Average Case):** زمان اجرای الگوریتم وقتی که یکی از همه‌ی ورودی‌های ممکن با اندازه‌ی  $n$  به طور شانسی (با احتمال یکسان) به برنامه داده شود. یا به عبارت دیگر، متوسط زمان اجرای برنامه برای همه‌ی ورودی‌های به اندازه‌ی  $n$ .

# حالات میانگین، بهترین و بدترین

---

1. بهترین حالت (Best Case): حد پایینی از زمان اجرا

2. بدترین حالت (Worst Case): حد بالایی از زمان اجرا

3. حالت میانگین (Average Case): زمان اجرا به ازای ورودی تصادفی (Random)

**نکته:** اگر رفتار یک الگوریتم یا پیچیدگی آن در بهترین و بدترین حالات مختلف باشد، ممکن است پیچیدگی الگوریتم در حالت میانگین بهتر از پیچیدگی آن در بدترین حالت باشد.



## نکته مهم

- ❖ اگر رفتار یک الگوریتم یا پیچیدگی آن در بهترین و بدترین حالات مختلف باشد، ممکن است پیچیدگی الگوریتم در حالت میانگین بهتر از پیچیدگی آن در بدترین حالت باشد.
- ❖ محاسبه حالت میانگین اجرا، سخت تر از محاسبه بدترین حالت: زیرا نیاز به داشتن **دانش قبلی** در مورد تمام داده های ورودی دارد. به عنوان مثال در مسئله مرتب سازی چند درصد از ارایه ورودی مرتب است و توزیع داده ها چگونه است.
- ❖ اگر برای محاسبه مدت زمان اجرا برای حالت میانگین، متوسط بهترین و بدترین گرفته شود، **اشتباه** است
- ❖ زیرا کاملاً به برنامه ما بستگی دارد. ممکن است برنامه ما برای  $(1 - 1/n) \%$  موقع در بهترین حالت با درجه  $n$  و در  $1/n \%$  موقع در بدترین حالت با مرتبه  $n^2$  باشد. بنابراین برنامه ما از درجه  $n$  برای حالت متوسط خواهد بود در حالی که همچنان بدترین مرتبه اجرا  $n^2$  است
- ❖ به همین دلایل، معمولاً بدترین حالت در نظر گرفته می شود تا مطمین شویم مدت زمان اجرای برنامه دیگه از این بیشتر نمیشه

## مقایسه توابع رشد

- ❖ می‌توان از جمله‌ها با درجه کوچکتر از بزرگترین درجه چشم پوشی کرد.
- ❖ در  $N$ ‌های کوچک معمولاً مقدار توابع کم است و برای ما مهم نیست.
- ❖ در  $N$ ‌های بزرگ تاثیر خیلی کمی روی تابع می‌گذارند.

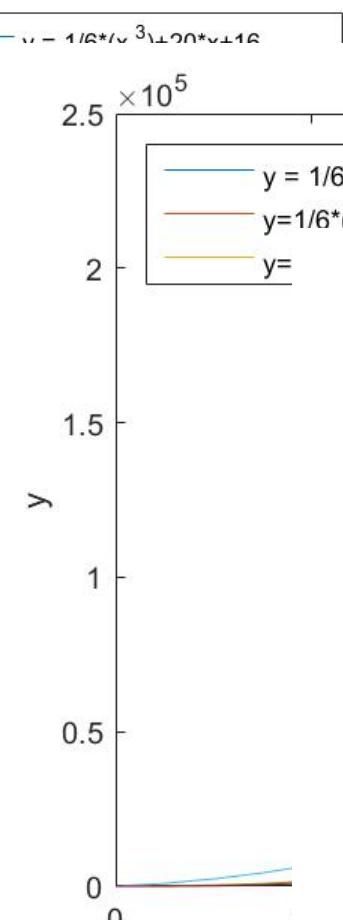
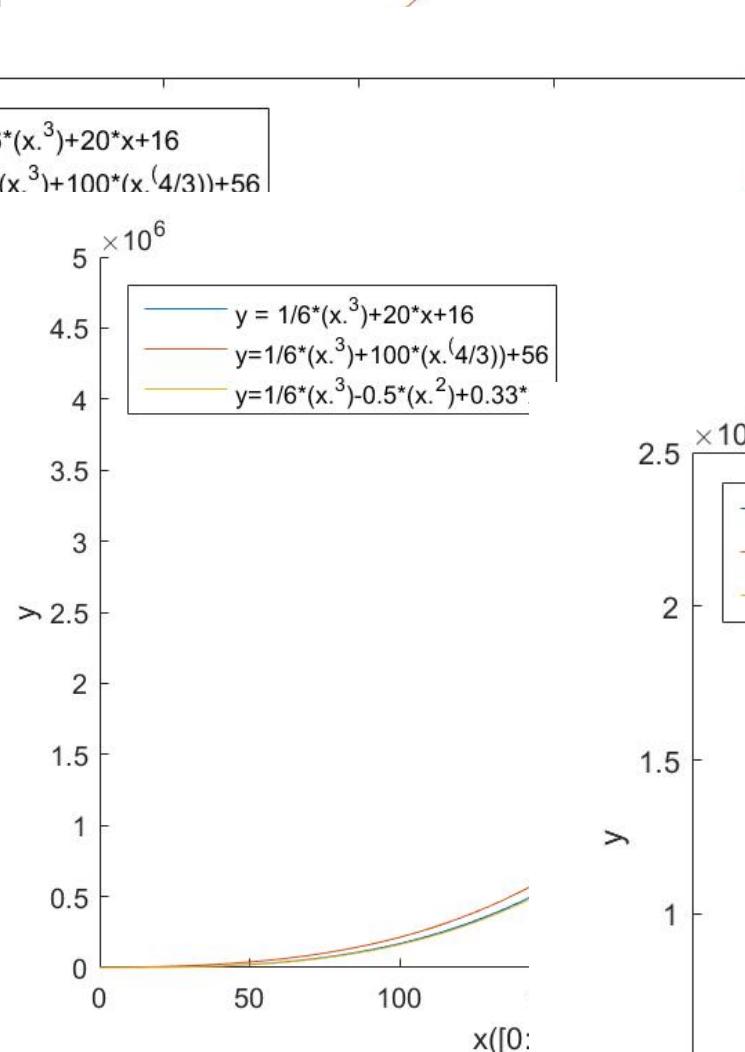
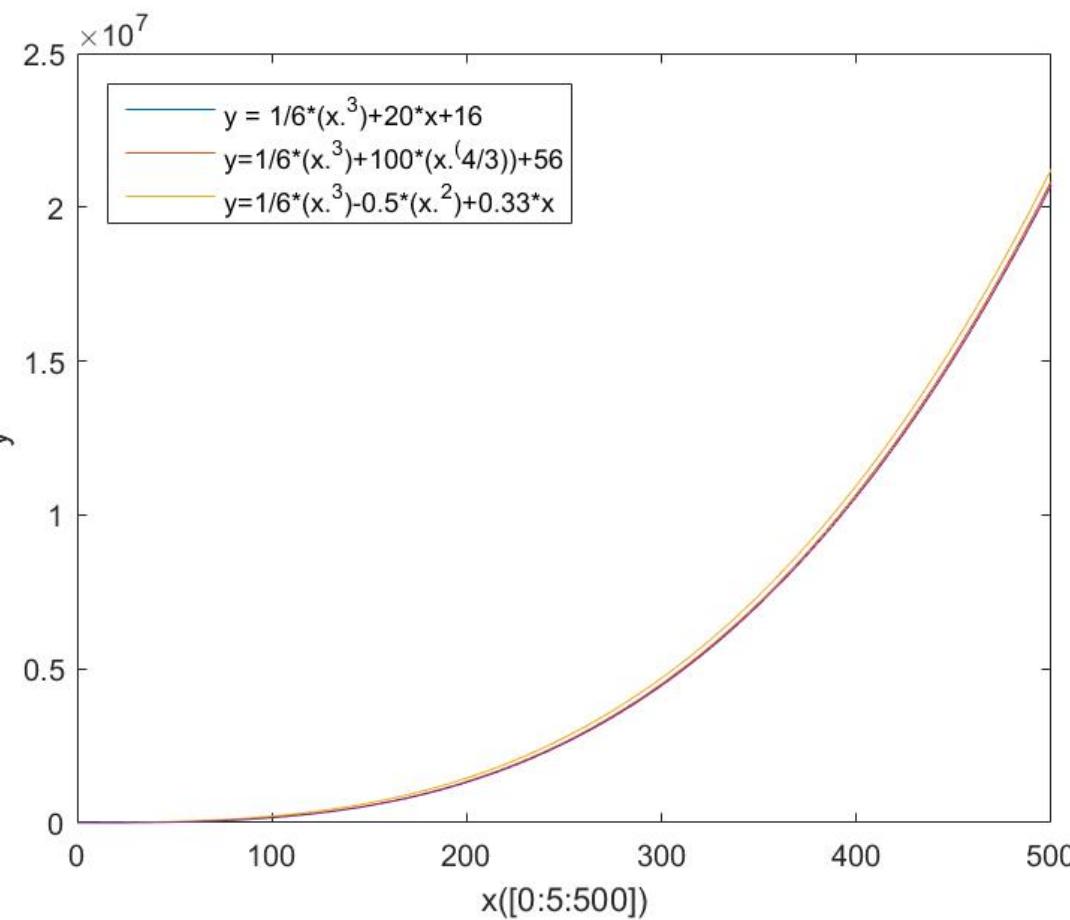
$$\frac{1}{6}N^3 + 20N + 16 \sim \frac{1}{6}N^3$$

$$\frac{1}{6}N^3 + 100N^{4/3} + 56 \sim \frac{1}{6}N^3$$

$$\frac{1}{6}N^3 - \frac{1}{2}N^2 + \frac{1}{3}N \sim \frac{1}{6}N^3$$

چرا از جمله ها ب

د



## مقایسه توابع رشد

---

به عبارت دیگر در این مثال‌ها هر دو تابع  $f$  و  $g$  روند رشد یکسانی دارند چرا که:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1 \rightarrow f(n) \sim g(n)$$

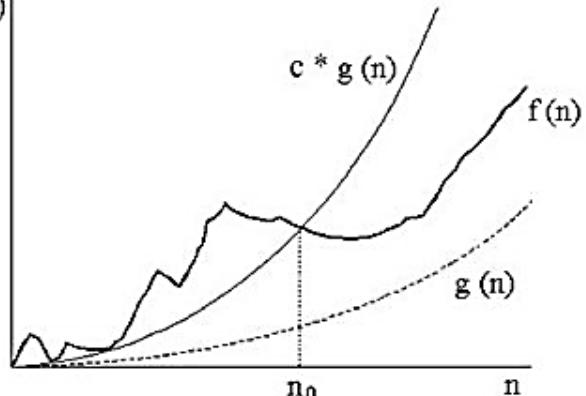
حال می‌توان گفت که در مقایسه‌ی زمان الگوریتم‌ها بین دو الگوریتم از یک دسته تفاوت چندانی وجود ندارد و دو الگوریتم زمانی به طور قابل توجه از لحاظ زمانی متفاوت هستند که از دو دسته‌ی متفاوت باشند.

# عبارت های مورد استفاده برای تحلیل پیچیدگی

---

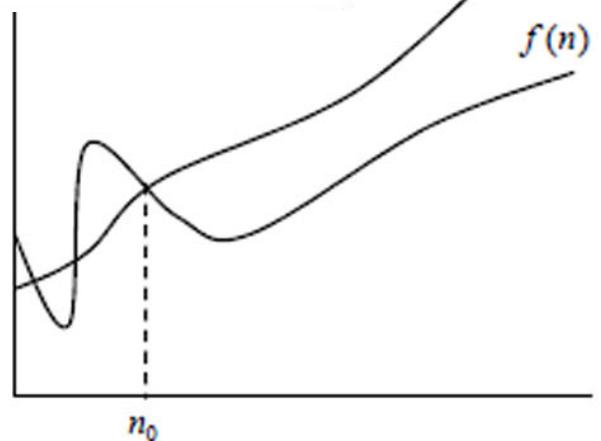
## Asymptotic Notations $\Theta$ , $O$ , $\Omega$ , $o$ , $\omega$

- We use  $\Theta$  to mean “order exactly”,
- $O$  to mean “order at most”,
- $\Omega$  to mean “order at least”,
- $o$  to mean “tight upper bound”,
- $\omega$  to mean “tight lower bound”,

$T(n)$ 

# The (Big) O Notation

$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$



$g(n)$  is an asymptotic upper bound for  $f(n)$ .

## Examples:

$$n^2 = O(n^2)$$

$$n = O(n^2)$$

$$n^2 + n = O(n^2)$$

$$\frac{n}{1200} = O(n^2)$$

$$n^2 + 1000n = O(n^2)$$

$$n^{1.99999} = O(n^2)$$

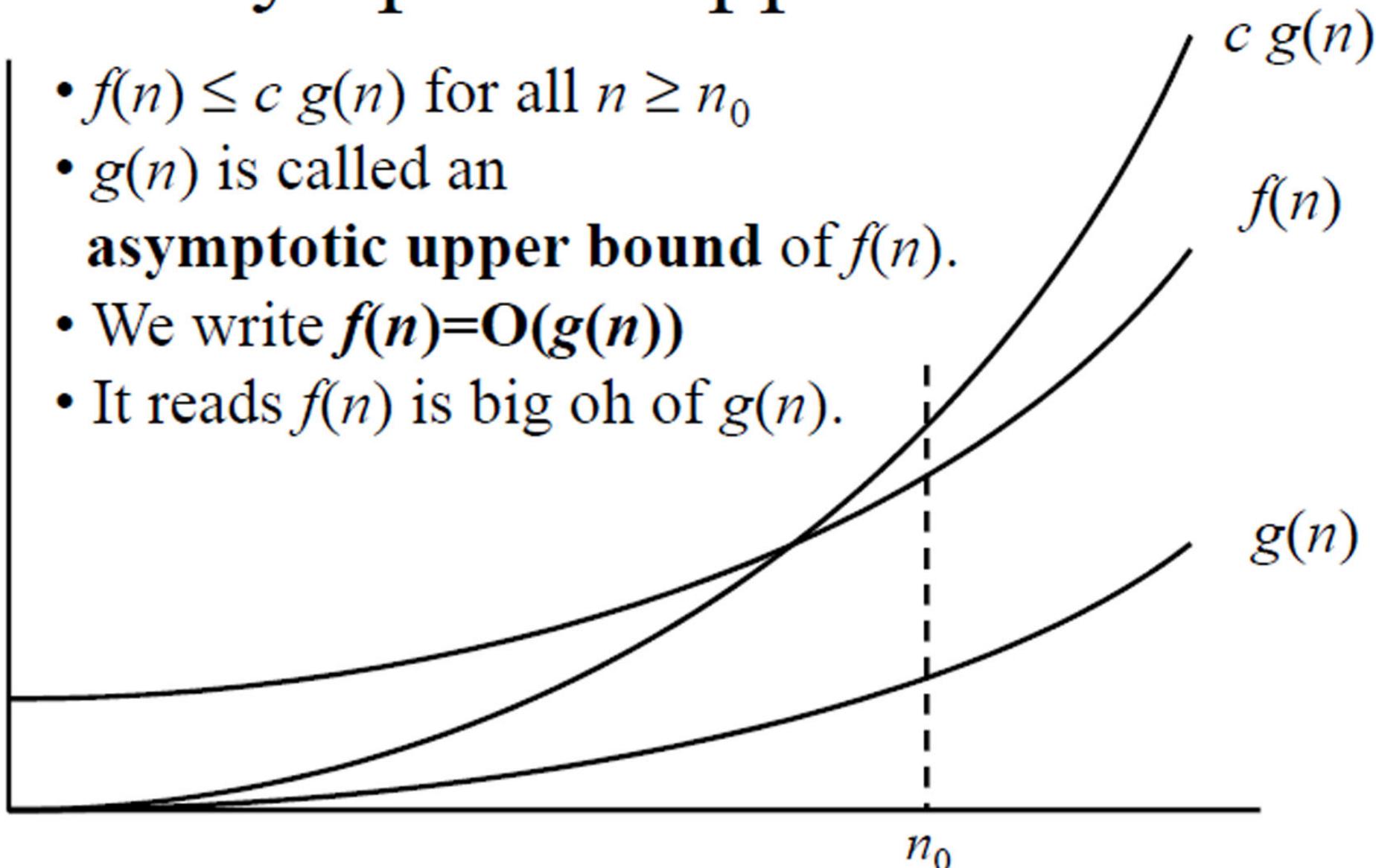
$$5230n^2 + 1000n = O(n^2)$$

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

**Note:** Since changing the base of a log only changes the function by a constant factor, we usually don't worry about log bases in asymptotic notation.

# Asymptotic Upper Bound

- $f(n) \leq c g(n)$  for all  $n \geq n_0$
- $g(n)$  is called an **asymptotic upper bound** of  $f(n)$ .
- We write  $f(n)=O(g(n))$
- It reads  $f(n)$  is big oh of  $g(n)$ .



# Uses of Big O

$$5n + 34\lg n + 2 = 5n + O(\lg n)$$

You can use Big O to describe low-order terms that you want to leave in the equation.

$$\left. \begin{array}{l} n^2 + O(n \lg n) \\ O(n^2) + n \lg n = O(n^2) \end{array} \right\}$$

If you leave a high-order term outside Big O, you must keep it. If the high-order term can be expressed by your Big O functions, you can drop low-order terms.

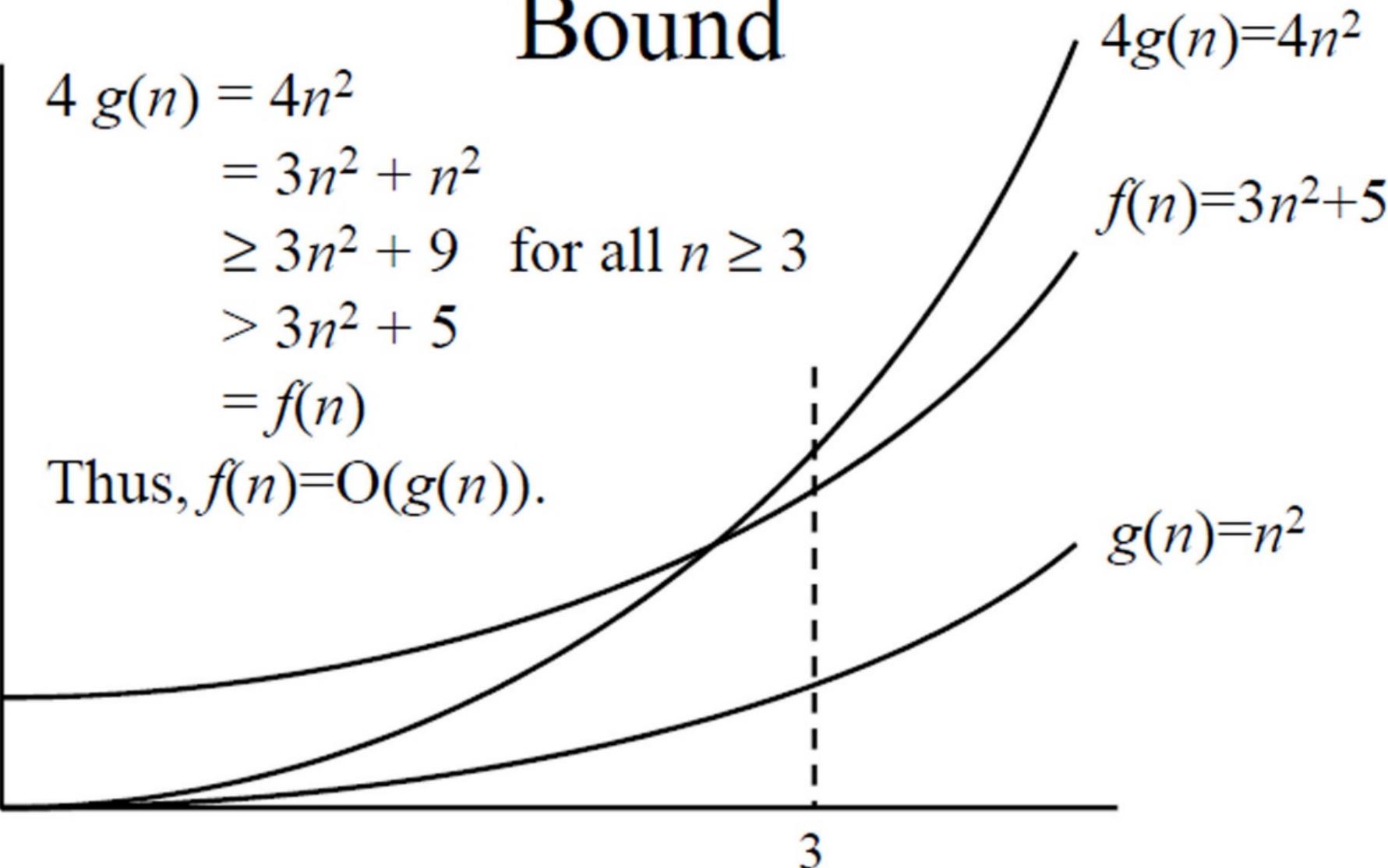
$$O(n^2) * O(\lg n) * O(n) = O(n^3 \lg n)$$

When you multiply terms using Big O, carry out the operations as normal, but at the end, bring the products with Big O **into** Big O.

## Example of Asymptotic Upper Bound

- $$\begin{aligned}4g(n) &= 4n^2 \\&= 3n^2 + n^2 \\&\geq 3n^2 + 9 \quad \text{for all } n \geq 3 \\&> 3n^2 + 5 \\&= f(n)\end{aligned}$$

Thus,  $f(n)=O(g(n))$ .



**Example 1:** Prove that  $2n^2 \in O(n^3)$

**Proof:**

Assume that  $f(n) = 2n^2$ , and  $g(n) = n^3$

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

Now we have to find the existence of  $c$  and  $n_0$

$$f(n) \leq c.g(n) \Leftrightarrow 2n^2 \leq c.n^3 \Leftrightarrow 2 \leq c.n$$

if we take,  $c = 1$  and  $n_0 = 2$  OR

$c = 2$  and  $n_0 = 1$  then

$$2n^2 \leq c.n^3$$

Hence  $f(n) \in O(g(n))$ ,  $c = 1$  and  $n_0 = 2$

## Example 2: Prove that $n^2 \in O(n^2)$

Proof:

Assume that  $f(n) = n^2$ , and  $g(n) = n^2$

Now we have to show that  $f(n) \in O(g(n))$

Since

$f(n) \leq c.g(n) \Leftrightarrow n^2 \leq c.n^2 \Leftrightarrow 1 \leq c$ , take,  $c = 1$ ,  $n_0 = 1$

Then

$n^2 \leq c.n^2$  for  $c = 1$  and  $n \geq 1$

Hence,  $2n^2 \in O(n^2)$ , where  $c = 1$  and  $n_0 = 1$

**Example 3: Prove that  $1000 \cdot n^2 + 1000 \cdot n \in O(n^2)$**

**Proof:**

Assume that  $f(n) = 1000 \cdot n^2 + 1000 \cdot n$ , and  $g(n) = n^2$

We have to find existence of  $c$  and  $n_0$  such that

$$0 \leq f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$

$$1000 \cdot n^2 + 1000 \cdot n \leq c \cdot n^2 = 1001 \cdot n^2, \text{ for } c = 1001$$

$$1000 \cdot n^2 + 1000 \cdot n \leq 1001 \cdot n^2$$

$$\Leftrightarrow 1000 \cdot n \leq n^2 \Leftrightarrow n^2 \geq 1000 \cdot n \Leftrightarrow n^2 - 1000 \cdot n \geq 0$$

$$\Leftrightarrow n(n-1000) \geq 0, \text{ this true for } n \geq 1000$$

$$f(n) \leq c \cdot g(n) \quad \forall n \geq n_0 \text{ and } c = 1001$$

Hence  $f(n) \in O(g(n))$  for  $c = 1001$  and  $n_0 = 1000$

## Example 4: Prove that $n^3 \not\in O(n^2)$

Proof:

- On contrary we assume that there exist some positive constants  $c$  and  $n_0$  such that

$$0 \leq n^3 \leq c \cdot n^2 \quad \heartsuit \quad n \geq n_0$$

$$0 \leq n^3 \leq c \cdot n^2 \Leftarrow n \leq c$$

در این حالت مقدار  $c$  به  
بستگی دارد و ثابت نیست

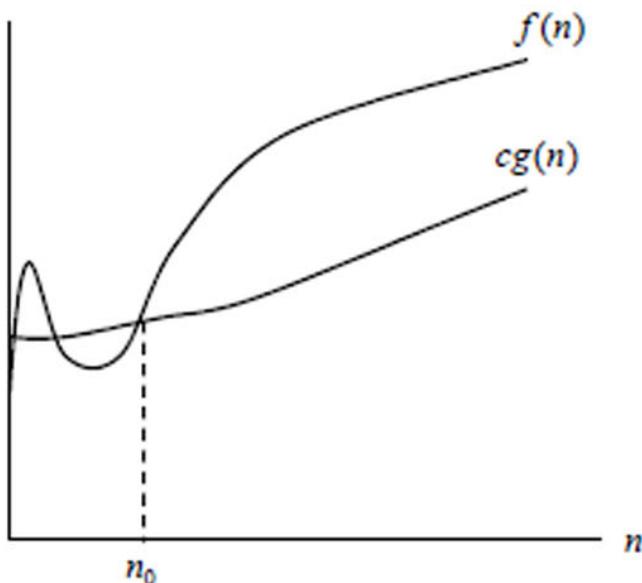
Since  $c$  is any fixed number and  $n$  is any arbitrary constant, therefore  $n \leq c$  is not possible in general.

Hence our supposition is wrong and  $n^3 \leq c \cdot n^2$ ,

$\heartsuit n \geq n_0$  is not true for any combination of  $c$  and  $n_0$ .

And hence,  $n^3 \not\in O(n^2)$

# The $\Omega$ Notation



$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$   
 $0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$

$g(n)$  is an asymptotic lower bound for  $f(n)$ .

## Examples:

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

$$n^{2.0001} = \Omega(n^2)$$

$$n^2 + n = \Omega(n^2)$$

$$n^2 \lg \lg n = \Omega(n^2)$$

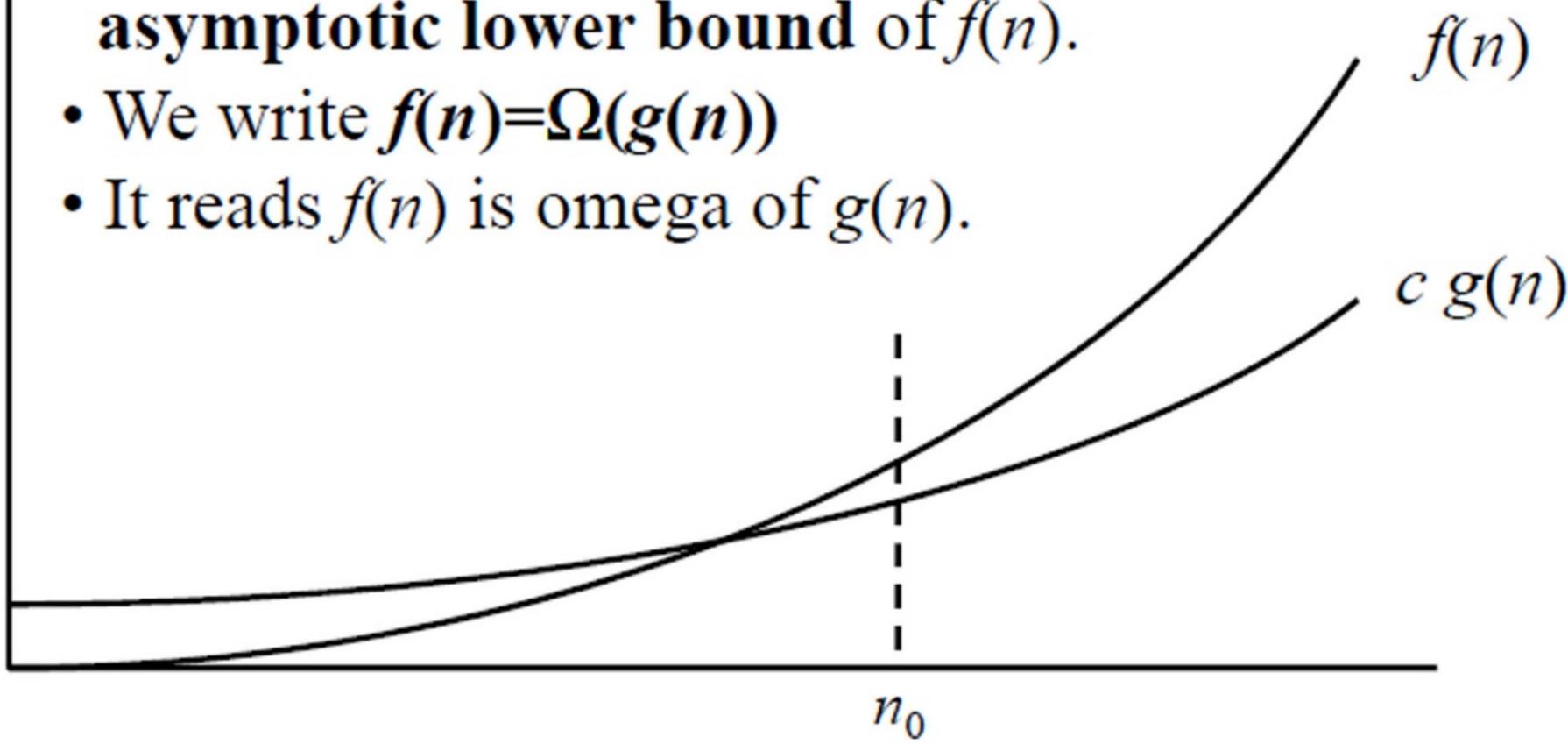
$$2311n^2 + 1000n = \Omega(n^2)$$

$$n^3 = \Omega(n^2)$$

$$2^{2^n} = \Omega(n^2)$$

# Asymptotic Lower Bound

- $f(n) \geq c g(n)$  for all  $n \geq n_0$
- $g(n)$  is called an **asymptotic lower bound** of  $f(n)$ .
- We write  $f(n)=\Omega(g(n))$
- It reads  $f(n)$  is omega of  $g(n)$ .



# Example of Asymptotic Lower Bound

$$g(n)/4 = n^2/4$$

$$= n^2/2 - n^2/4$$

$$\leq n^2/2 - 9 \text{ for all } n \geq 6$$

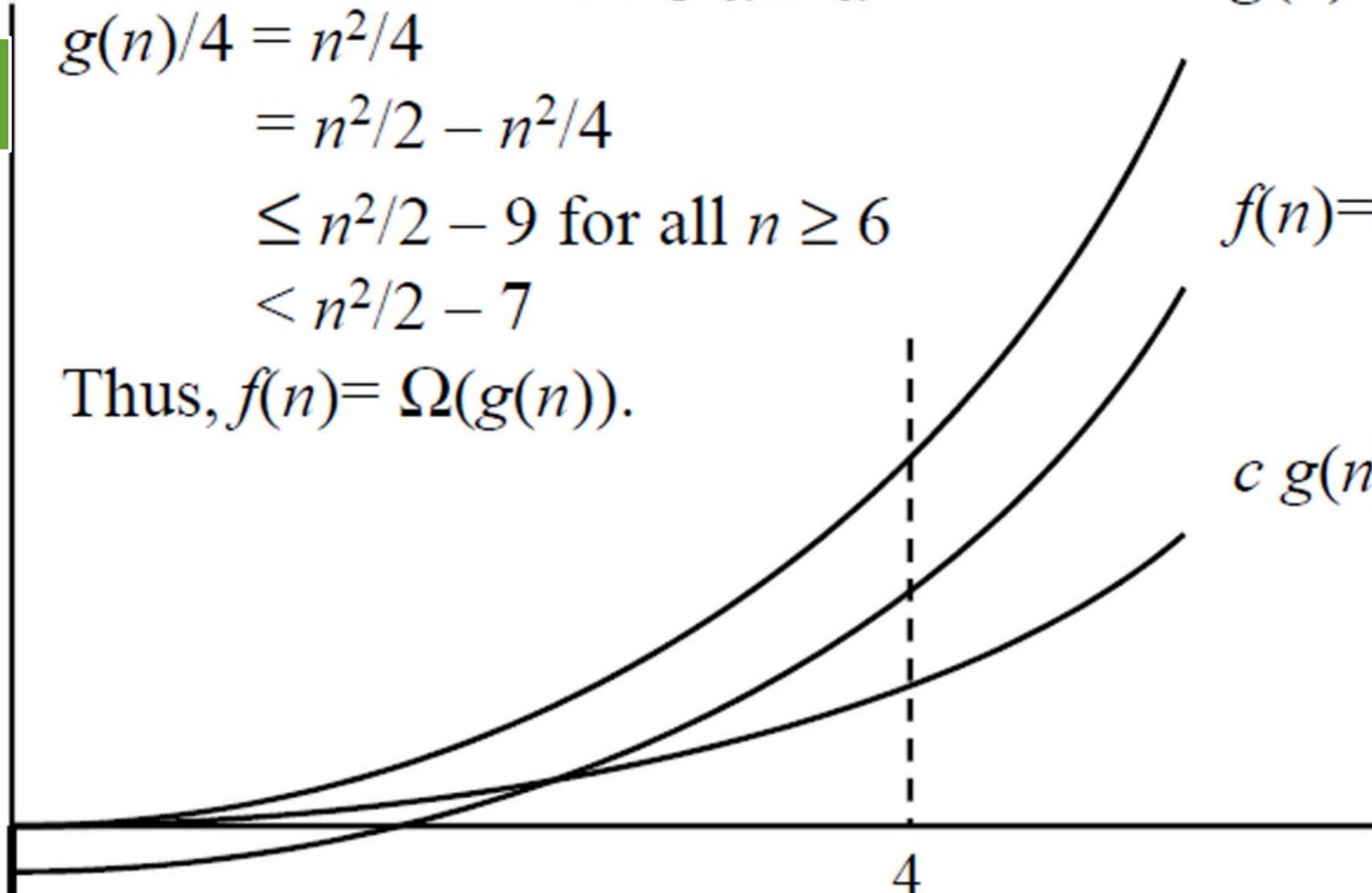
$$< n^2/2 - 7$$

Thus,  $f(n) = \Omega(g(n))$ .

$$g(n)=n^2$$

$$f(n)=n^2/2-7$$

$$c g(n)=n^2/16$$



Example 1: Prove that  $5.n^2 \in \Omega(n)$

Proof:

Assume that  $f(n) = 5.n^2$ , and  $g(n) = n$

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

We have to find the existence of  $c$  and  $n_0$  s.t.

$$c.g(n) \leq f(n) \quad \forall n \geq n_0$$

$$c.n \leq 5.n^2 \Leftrightarrow c \leq 5.n$$

if we take,  $c = 5$  and  $n_0 = 1$  then

$$c.n \leq 5.n^2 \quad \forall n \geq n_0$$

And hence  $f(n) \in \Omega(g(n))$ , for  $c = 5$  and  $n_0 = 1$

## Example 2: Prove that $5.n + 10 \in \Omega(n)$

Proof:

- Assume that  $f(n) = 5.n + 10$ , and  $g(n) = n$   
 $f(n) \in \Omega(g(n))$  ?

We have to find the existence of  $c$  and  $n_0$  s.t.

$$c.g(n) \leq f(n) \quad \forall n \geq n_0$$

$$c.n \leq 5.n + 10 \Leftrightarrow c.n \leq 5.n + 10.n \Leftrightarrow c \leq 15.n$$

if we take,  $c = 15$  and  $n_0 = 1$  then

$$c.n \leq 5.n + 10 \quad \forall n \geq n_0$$

And hence  $f(n) \in \Omega(g(n))$ , for  $c = 15$  and  $n_0 = 1$

## Example 3: Prove that $100.n + 5 \notin \Omega(n^2)$

Proof:

Let  $f(n) = 100.n + 5$ , and  $g(n) = n^2$

Assume that  $f(n) \in \Omega(g(n))$  ?

Now if  $f(n) \in \Omega(g(n))$  then there exist  $c > 0$  and  $n_0 \geq 0$  such that

$$c.g(n) \leq f(n) \quad \forall n \geq n_0 \quad \Leftarrow$$

$$c.n^2 \leq 100.n + 5 \quad \Leftarrow$$

$$c.n \leq 100 + 5/n \quad \Leftarrow$$

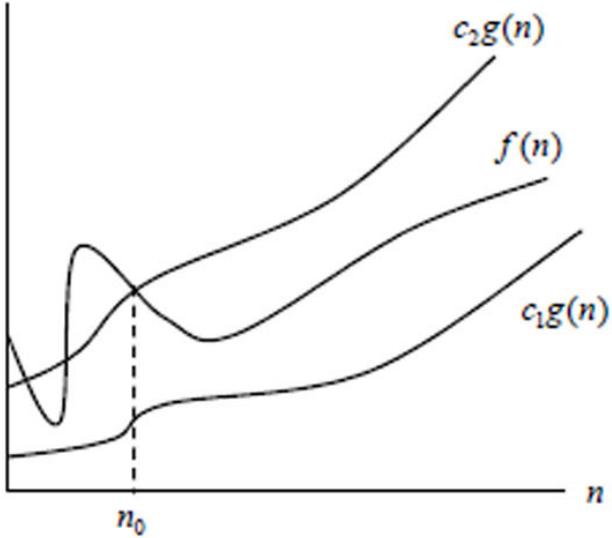
$n \leq 100/c$ , for a very large  $n$ , which is not possible

And hence  $f(n) \not\in \Omega(g(n))$

با میل  $n$  به سمت بی نهایت، عبارت  $n/5$  به صفر میل میکند. از طرف دیگر مقدار حد بالای  $n$  برابر با  $c/100$  میشود که با فرض میل  $n$  به بی نهایت متناقض است

# The $\Theta$ Notation

$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\}.$



$g(n)$  is an asymptotically tight bound for  $f(n)$ .

## Example:

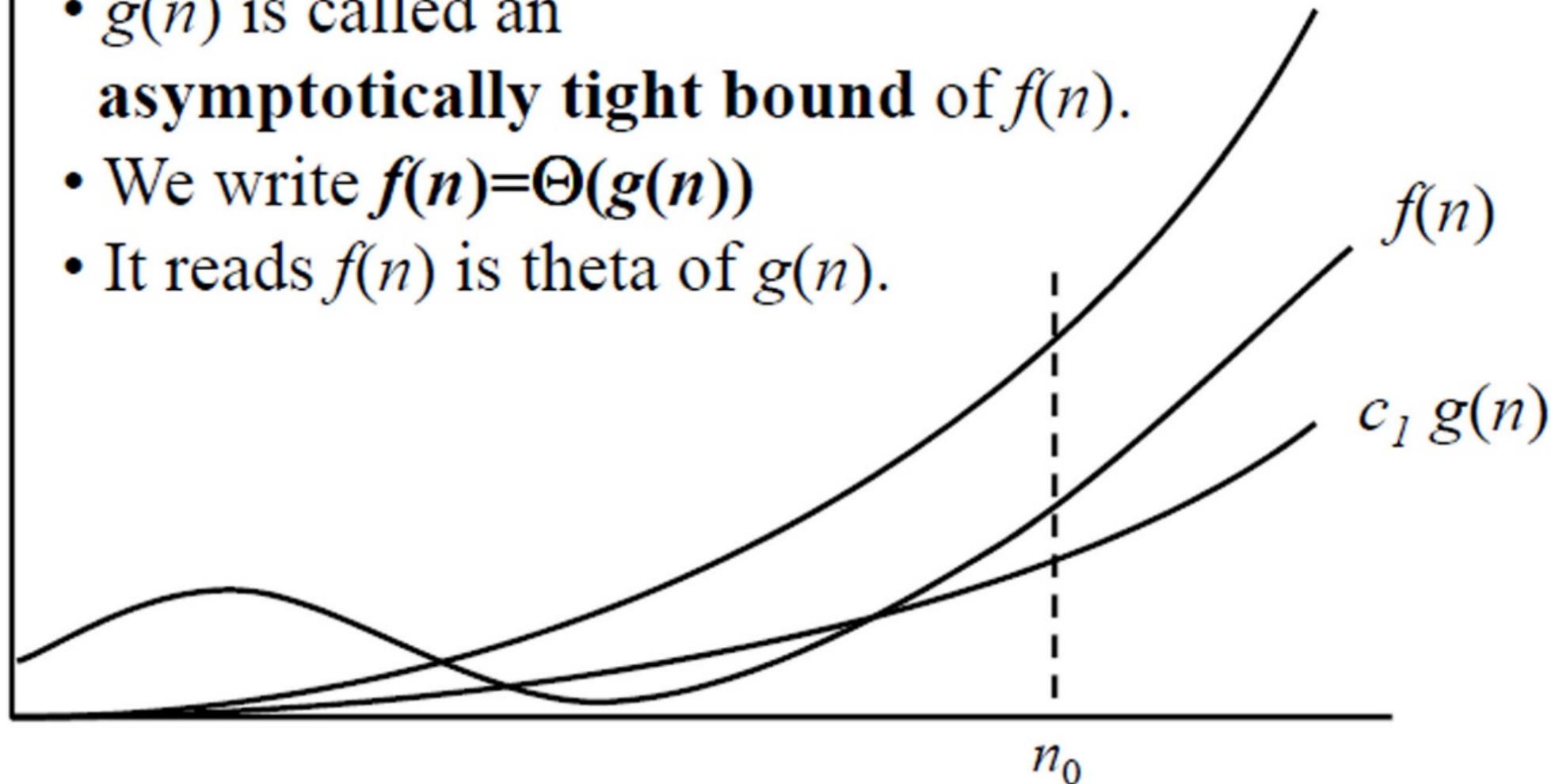
$$\frac{n^2}{2} - 2n = \Theta(n^2), \text{ with } c_1 = \frac{1}{4}, c_2 = \frac{1}{2}, \text{ and } n_0 = 8.$$

# Asymptotically Tight Bound

- $f(n) = O(g(n))$  and  $f(n) = \Omega(g(n))$

- $g(n)$  is called an **asymptotically tight bound** of  $f(n)$ .

- We write  $f(n)=\Theta(g(n))$
- It reads  $f(n)$  is theta of  $g(n)$ .



Example 1: Prove that  $\frac{1}{2}n^2 - \frac{1}{2}n = \Theta(n^2)$

### Proof

Assume that  $f(n) = \frac{1}{2}n^2 - \frac{1}{2}n$ , and  $g(n) = n^2$

$f(n) \in \Theta(g(n))$ ?

We have to find the existence of  $c_1, c_2$  and  $n_0$  s.t.

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \forall n \geq n_0$$

Since,  $\frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2$        $\forall n \geq 0$  if  $c_2 = \frac{1}{2}$  and

$$\frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n \cdot \frac{1}{2}n \quad (\forall n \geq 2) = \frac{1}{4}n^2, \quad c_1 = \frac{1}{4}$$

Hence  $\frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \leq \frac{1}{2}n^2 - \frac{1}{2}n$

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \forall n \geq 2, \quad c_1 = \frac{1}{4}, \quad c_2 = \frac{1}{2}$$

Hence  $f(n) \in \Theta(g(n)) \Rightarrow \frac{1}{2}n^2 - \frac{1}{2}n = \Theta(n^2)$

Example 1: Prove that  $2.n^2 + 3.n + 6 \not\in \Theta(n^3)$

( Proof: Let  $f(n) = 2.n^2 + 3.n + 6$ , and  $g(n) = n^3$

E we have to show that  $f(n) \not\in \Theta(g(n))$

On contrary assume that  $f(n) \in \Theta(g(n))$  i.e.

there exist some positive constants  $c_1, c_2$  and  $n_0$   
such that:  $c_1.g(n) \leq f(n) \leq c_2.g(n)$

$$c_1.g(n) \leq f(n) \leq c_2.g(n) \Leftrightarrow c_1.n^3 \leq 2.n^2 + 3.n + 6 \leq c_2.n^3 \Leftrightarrow$$

$$c_1.n \leq 2 + 3/n + 6/n^2 \leq c_2.n \Rightarrow$$

$$c_1.n \leq 2 \leq c_2.n, \text{ for large } n \Rightarrow$$

$n \leq 2/c_1 \leq c_2/c_1.n$  which is not possible

Hence  $f(n) \not\in \Theta(g(n)) \Rightarrow 2.n^2 + 3.n + 6 \not\in \Theta(n^3)$

## LITTLE-OH

$\text{o}$ -notation is used to denote a upper bound that is not asymptotically tight.

For a given function  $g(n) \geq 0$ , denoted by  $o(g(n))$  the set of functions,

$$o(g(n)) = \left\{ f(n) \mid \begin{array}{l} \text{for any positive constants } c, \text{ there exists a constant } n_o \\ \text{such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_o \end{array} \right\}$$

$f(n)$  becomes insignificant relative to  $g(n)$  as  $n$  approaches infinity

e.g.,  $2n = o(n^2)$  but  $2n^2 \neq o(n^2)$ .  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

$g(n)$  is an upper bound for  $f(n)$ , not asymptotically tight

**Example 1:** Prove that  $2n^2 \in o(n^3)$

**Proof:**

Assume that  $f(n) = 2n^2$ , and  $g(n) = n^3$   
 $f(n) \in o(g(n))$  ?

Now we have to find the existence  $n_0$  for any  $c$   
 $f(n) < c.g(n)$  this is true

$$\Leftrightarrow 2n^2 < c.n^3 \Leftrightarrow 2 < c.n$$

This is true for any  $c$ , because for any arbitrary  $c$   
we can choose  $n_0$  such that the above inequality  
holds.

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

**Example 3:** Prove that  $1000.n^2 + 1000.n \notin o(n^2)$

**Proof:**

— Assume that  $f(n) = 1000.n^2 + 1000.n$ , and  $g(n) = n^2$   
we have to show that  $f(n) \notin o(g(n))$  i.e.

We assume that for any  $c$  there exist  $n_0$  such that

$$0 \leq f(n) < c.g(n) \quad \forall n \geq n_0$$

$$1000.n^2 + 1000.n < c.n^2$$

با انتخاب  $c=1000$ ، مقدار  $n > 0$  می شود، در نتیجه  
برای هر  $c$  ای برقرار نخواهد بود

# LITTLE-OMEGA

- Little- $\omega$  notation is used to denote a lower bound that is not asymptotically tight.

For a given function  $g(n)$ , denote by  $\omega(g(n))$  the set of all functions.

$\omega(g(n)) = \{f(n) \mid \text{for any positive constants } c, \text{ there exists a constant } n_o \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_o\}$

$f(n)$  becomes arbitrarily large relative to  $g(n)$  as  $n$  approaches infinity

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

e.g.,  $\frac{n^2}{2} = \omega(n)$  but  $\frac{n^2}{2} \neq \omega(n^2)$ .

**Example 1:** Prove that  $5.n^2 \in \omega(n)$

**Proof:**

Assume that  $f(n) = 5.n^2$ , and  $g(n) = n$   
 $f(n) \in \Omega(g(n))$  ?

We have to prove that for any  $c$  there exists  $n_0$  s.t.,  
 $c.g(n) < f(n) \quad \forall n \geq n_0$   
 $c.n < 5.n^2 \Leftrightarrow c < 5.n$

This is true for any  $c$ , because for any arbitrary  $c$   
e.g.  $c = 1000000$ , we can choose  $n_0 = 1000000/5$   
 $= 200000$  and the above inequality does hold.

And hence  $f(n) \in \omega(g(n))$ ,

Example 2: Prove that  $5.n + 10 \notin \omega(n)$

Proof:

Assume that  $f(n) = 5.n + 10$ , and  $g(n) = n$   
 $f(n) \notin \Omega(g(n))$  ?

We have to find the existence  $n_0$  for any  $c$ , s.t.

$$c.g(n) < f(n) \quad \forall n \geq n_0$$

$c.n < 5.n + 10$ , if we take  $c = 16$  then

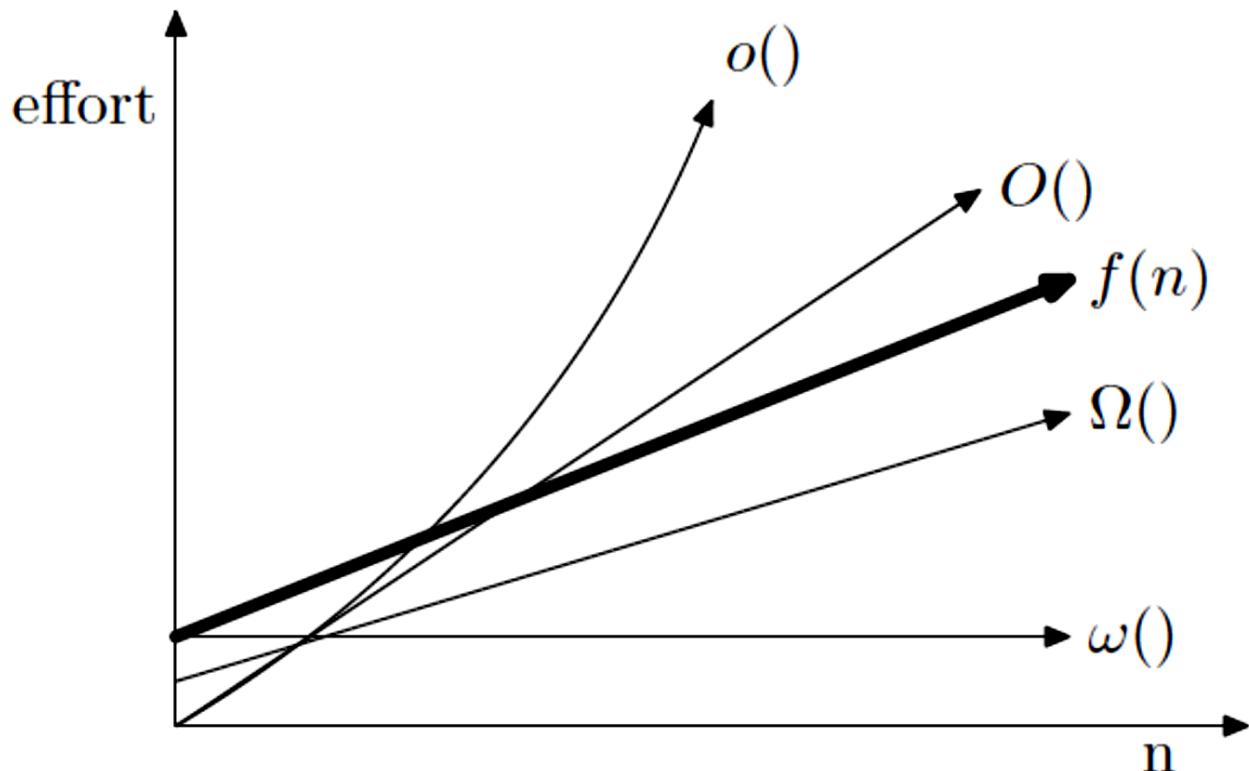
$16.n < 5.n + 10 \Leftrightarrow 11.n < 10$  is not true for any positive integer.

Hence  $f(n) \notin \omega(g(n))$

# حمع بندی

نتیجه‌ای که از تعریف‌های فوق به دست می‌آید را در عبارت‌های زیر (هر چند نادقيق از نظر ریاضی) :

اگر  $F$  درجه‌ی



# جمع بندی

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

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \begin{cases} \rightarrow 1 \\ \rightarrow 0 \end{cases}$$

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

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

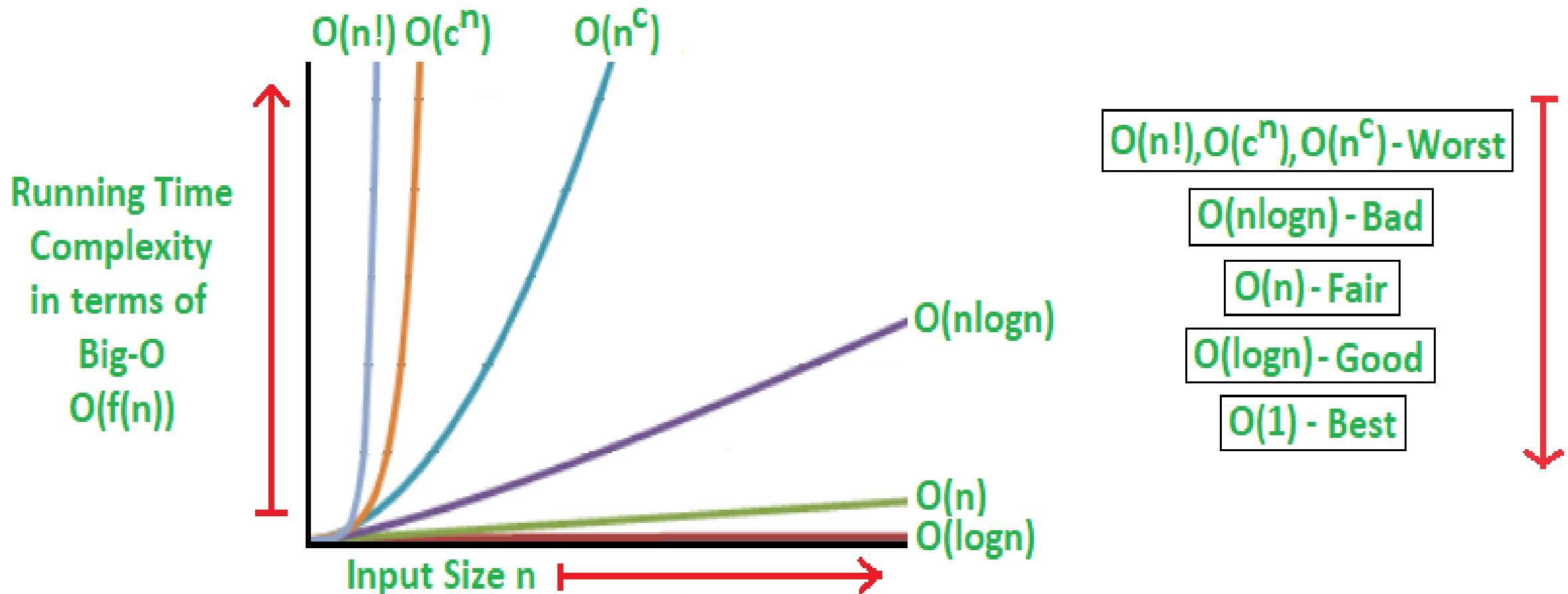
$$f(n) \in \Omega(g(n))$$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \begin{cases} \rightarrow 1 \\ \rightarrow \infty \end{cases}$$

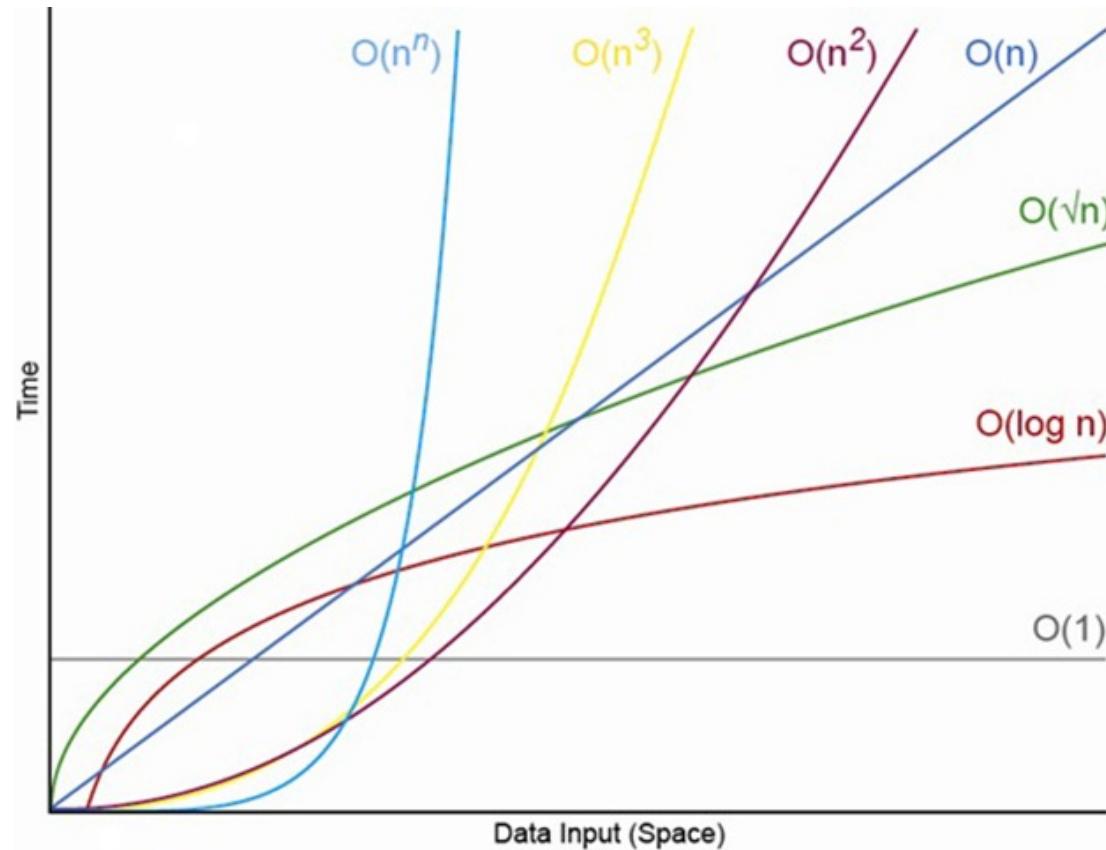
$$f(n) \in \omega(g(n))$$

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

# A COMPARISON OF GROWTH-RATE FUNCTIONS...



# A COMPARISON OF GROWTH-RATE FUNCTIONS...



# پیچیدگی الگوریتم ها

❖ زمان اجرای الگوریتم ها به ازای مقادیر مختلف  $n$ :

class	$n=2$	$n=16$	$n=256$	$n=1024$
$1$	$1$	$1$	$1$	$1$
$\log(n)$	$1$	$4$	$8$	$10$
$n$	$2$	$16$	$256$	$1024$
$n\log(n)$	$2$	$64$	$2048$	$10240$
$n^2$	$4$	$256$	$65536$	$1048576$
$n^3$	$8$	$4096$	$16777216$	$1.07E+09$
$2^n$	$4$	$65536$	$1.16E+77$	$1.8E+308$

# مقایسه نرخ رشد توابع مختلف از نظر زمان اجرا

بزرگ‌ترین مسئله‌ی قابل حل در چند دقیقه				نرخ رشد
۲۰۰۰ به بعد	۹۰ دهه‌ی	۸۰ دهه‌ی	۷۰ دهه‌ی	
هر	هر	هر	هر	۱
هر	هر	هر	هر	$\log N$
چند میلیارد	چند صد میلیون	چند ده میلیون	چند میلیون	$N$
چند صد میلیون	چند ده میلیون	چند میلیون	چند صد هزار	$N \log N$
چند ده هزار	چند هزار	هزار	چند صد	$N^2$
چند هزار	هزار	چند صد	صد	$N^3$
سی	بیست و چند	بیست و چند	بیست	$2^N$

نتیجه. برای عقب نماندن از قاعده‌ی مور نیاز به الگوریتم‌های خطی یا خطی-لگاریتمی داریم.

# مقایسه نرخ رشد توابع مختلف از نظر زمان اجرا

$n f(n)$	$\lg n$	$n$	$n \lg n$	$n^2$	$2^n$	$n!$
10	0.003 $\mu s$	0.01 $\mu s$	0.033 $\mu s$	0.1 $\mu s$	1 $\mu s$	3.63 ms
20	0.004 $\mu s$	0.02 $\mu s$	0.086 $\mu s$	0.4 $\mu s$	1 ms	77.1 years
30	0.005 $\mu s$	0.03 $\mu s$	0.147 $\mu s$	0.9 $\mu s$	1 sec	$8.4 \times 10^{15}$ yrs
40	0.005 $\mu s$	0.04 $\mu s$	0.213 $\mu s$	1.6 $\mu s$	18.3 min	
50	0.006 $\mu s$	0.05 $\mu s$	0.282 $\mu s$	2.5 $\mu s$	13 days	
100	0.007 $\mu s$	0.1 $\mu s$	0.644 $\mu s$	10 $\mu s$	$4 \times 10^{13}$ yrs	
1,000	0.010 $\mu s$	1.00 $\mu s$	9.966 $\mu s$	1 ms		
10,000	0.013 $\mu s$	10 $\mu s$	130 $\mu s$	100 ms		
100,000	0.017 $\mu s$	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 $\mu s$	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 $\mu s$	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 $\mu s$	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 $\mu s$	1 sec	29.90 sec	31.7 years		

$f(n)$	Approximate number of data items processed per:			
	1 minute	1 day	1 year	1 century
$n$	10	14,400	$5.3 \times 10^6$	$5.3 \times 10^8$
$n \log_{10} n$	10	4,000	$8.8 \times 10^5$	$6.7 \times 10^7$
$n^{1.5}$	10	$1.3 \times 10^3$	$6.5 \times 10^4$	$1.4 \times 10^6$
$n^2$	10	380	$7.3 \times 10^3$	$7.3 \times 10^4$
$n^3$	10	110	810	$3.7 \times 10^3$
$2^n$	10	20	29	35

## Beware Exponential Complexity!

- A linear,  $O(n)$ , algorithm processing 10 items per minute, can process  $1.4 \times 10^4$  items per day,  $5.3 \times 10^6$  items per year, and  $5.3 \times 10^8$  items per century.
- An exponential,  $O(2^n)$ , algorithm processing 10 items per minute, can process only 20 items per day and only 35 items per century...

# تحلیل برنامه بازگشتی

❖ برنامه بازگشتی: برنامه‌ای که برای حل یک مسئله بازگشتی استفاده می شود

❖ مسئله بازگشتی: مسئله‌ای که برای حل آن، نیاز به حل همان مسئله با همان شرایط با ابعاد کوچکتر داریم

❖ نکته : هر مسئله بازگشتی، دارای حداقل یک پایه بازگشت (مقداری که به ازای آن جواب مسئله بازگشتی تعیین می شود) است

## ۲- جایگزینی و تکرار-فاکتوریل

❖ تحلیل یک برنامه بازگشتی، منجر به حل یک رابطه بازگشتی

```
Factorial(n)
{
    if n == 0
        return 1
    else
        return n * Factorial(n-1)
```

$$T(n) = T(n-1) + 3 \quad \text{if } n > 0$$

$$T(0) = 1$$

$$T(n) = T(n-1) + 3$$

$$= T(n-2) + 6$$

$$= T(n-3) + 9$$

$$= T(n-k) + 3k$$

$$n-k = 0 \Rightarrow k = n$$

$$\Rightarrow T(n) = T(0) + 3n$$

$$= 3n$$

# سری فیبوناچی

Series=0 1 1 2 3 5 8

Fib(n)

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

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

$$T(0) = T(1) = 1$$

$$T(n - 2) \leq T(n - 1)$$

$$c = 4$$

$$T(n) \geq 2T(n - 2) + c$$

$$T(n) \geq 2(2T(n - 4) + c) + c$$

تمرین: حد بالای سری را  
نیز بدست بیاورد

$$T(n) \geq 2^{\frac{k}{2}}T(n - k) + (2^{\frac{k}{2}-1})c, n - k = 0, k = n$$

$$T(n) \geq 2^{\frac{n}{2}}(1 + c) - c$$

$$T(n) \in \Omega(c2^{\frac{n}{2}}) = \Omega(2^{\frac{n}{2}})$$

## نکته

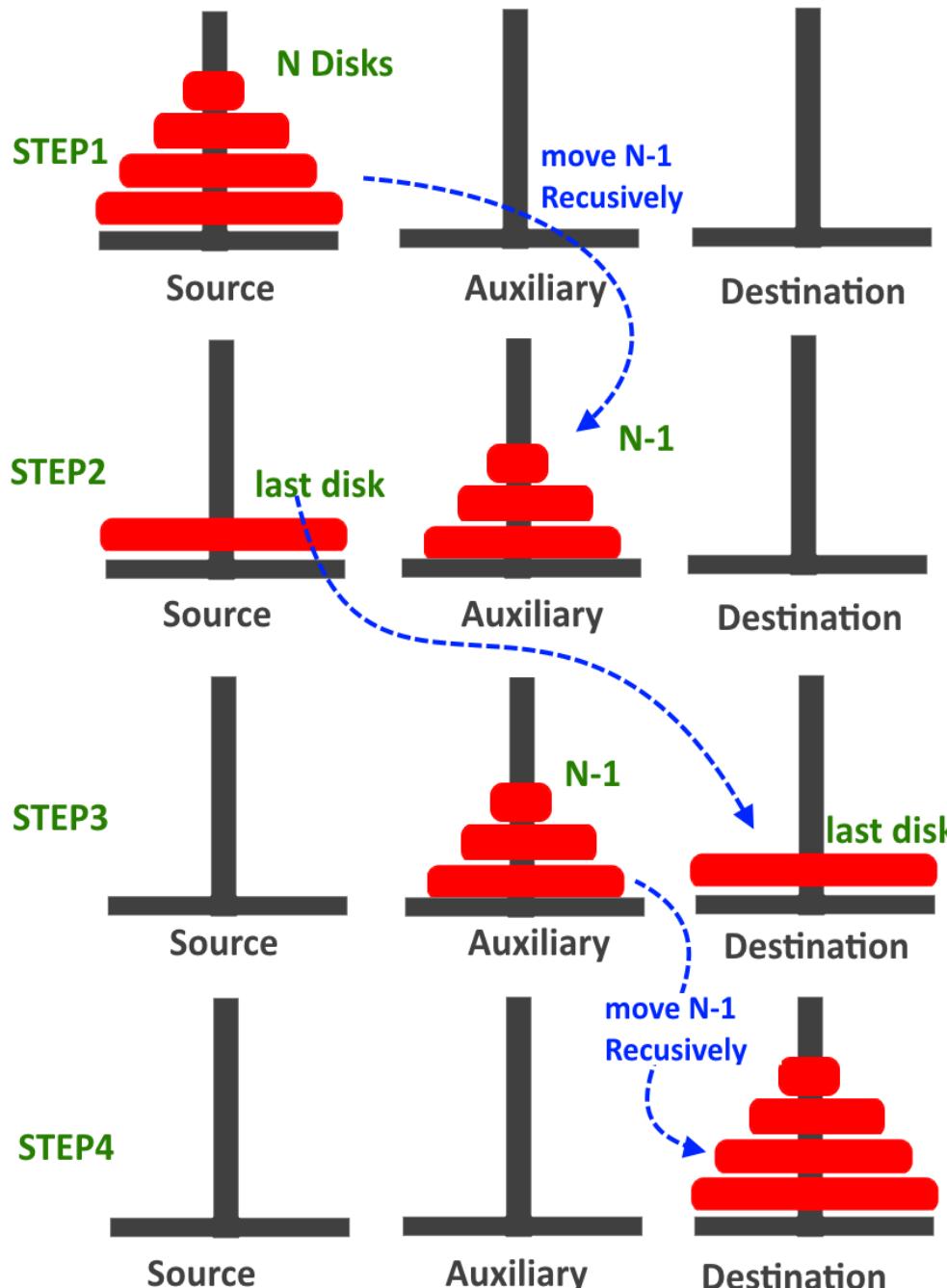
---

$$\begin{cases} T(n) = K T(n - 1) + C = O(K^n) & \forall K \geq 2 \\ T(n) = K T(n - B) + C = O\left(K^{\frac{n}{B}}\right) & \forall K \geq 2, B \geq 1 \end{cases}$$

$$T(n) = T(n - m) + O(g(n)) \Rightarrow T(n) = O(n * g(n))$$

Example:  $T(n) = T(n - 1) + n = O(n^2)$

# برج هانوی



قوانین:

- ۱- برای جابجایی دیاک ها فقط از میله ها می توان استفاده کرد
- ۲- در هر مرحله فقط ۱ دیاک را می توان جابجا کرد
- ۳- در مراحل میانی نباید هیچ دیاک بزرگتری روی دیاک کوچکتر از خودش قرار بگیرد

# برج هانوی

```
# Recursive Python function to solve tower of hanoi
```

```
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):  
    if n == 1:  
        print "Move disk 1 from rod",from_rod,"to rod",to_rod  
        return  
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)  
    print "Move disk",n,"from rod",from_rod,"to rod",to_rod  
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)
```

```
# Driver code
```

```
n = 4  
TowerOfHanoi(n, \'A\', \\'C\', \\'B\')  
# A, C, B are the name of rods
```

```
# Contributed By Harshit Agrawal
```

$$T(n) = \begin{cases} 1 & n = 1 \\ 2 T(n - 1) + 1 & n > 1 \end{cases}$$

$$O(2^n)$$

# برج هانوی

❖ برای  $n=64$  و اینکه فرض کنیم جایه جایی و انتقال هر دیاک فقط ۱ ثانیه طول بکشد، چقدر زمان برای انتقال این تعداد دیسک لازم است؟

❖ تقریباً ۶۰۰ میلیارد سال

$$❖ 2^{64} = 18,446,744,073,709,551,616$$

## تمرین

❖ فرض کنید مسئله برجهای هانوی را به سه میله A، B و C را در نظر بگیرید. فرض کنید که تمام شرایط هانوی برقرار است به علاوه این که هیچ حلقه ای را مستقیماً از A به C و یا از C به A منتقل کرد. به این معنی تنها انتقال ها به کمک میله B صورت میگیرد. اگر در ابتدا  $n$  حلقه در میله A باشد و  $T(n)$  تعداد عملیات لازم برای انتقال  $n$  حلقه از A به C باشد،  $T(n)$  چه مقدار است و کد آن را بنویسد.

## مثال - تعداد مقایسه ها در پیدا کردن کوچکترین و بزرگترین مقدار آرایه

```
if (arr[0] > arr[1])
{
    minmax.max = arr[0];
    minmax.min = arr[1];
}
else
{
    minmax.max = arr[1];
    minmax.min = arr[0];
}

for (i = 2; i<n; i++)
{
    if (arr[i] > minmax.max)
        minmax.max = arr[i];

    else if (arr[i] < minmax.min)
        minmax.min = arr[i];
}
```

بهترین حالت :  $1+n-2$

بدترین حالت :  $1+2(n-2)$

## MAXMIN( $A, p, q$ )

- ▷  $A$ : array,  $p, q$ : the first and last indices
- ▷ will return the indices of both max and min

```
1 if p=q
2   then max ← min ← A[p]
3 else if q - p = 1
4   then if A[p] > A[q]
5     then max ← A[p] ; min ← A[q]
6     else max ← A[q] ; min ← A[p]
7 else r ← ⌊ $\frac{p+q}{2}$ ⌋
8   min1,max1 ← MAXMIN( $A, p, r$ )
9   min2,max2 ← MAXMIN( $A, r + 1, q$ )
10  if max2 > max1
11    then max ← max2
12    else max ← max1
13  if min2 < min1
14    then min ← min2
15    else min ← min1
```

---

$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 2 & n > 2 \end{cases}$$

## آیا بھینه است؟

- 
- ❖ میدانیم که بهترین الگوریتم دارای  $2 - \left\lceil \frac{3n}{2} \right\rceil$  تعداد مقایسه است
  - ❖ در راه حل ارائه شده، مقدار  $T(6) = 8$  است در حالی که در راه بھینه این مقدار برابر ۷ است.
  - ❖ این راه برای موافقی که  $n = 2^k$  است، بھینه است (تمرین: با استفاده از تکرار و جایگزاری نشان دهید)

## راه برهینه ۱

- 
- ❖ یک مقایسه بین هر دو عنصر متوالی،  $\text{Min}$  ها و  $\text{Max}$  های هر زوج را پیدا کن
  - ❖ کوچکترین عنصر  $\text{Min}$  ها و حداقل یک عنصر اضافی (یعنی وقتی تعداد ورودی ها فرد باشد)، کوچکترین عنصر است
  - ❖ بزرگترین عنصر  $\text{Max}$  ها و حداقل یک عنصر اضافی (یعنی وقتی تعداد ورودی ها فرد باشد)، بزرگترین عنصر است

# راه برهینه ۱

$$n = 2k \quad \text{اگر} \diamond$$

$$\frac{n}{2} + \left(\frac{n}{2} - 1\right) + \left(\frac{n}{2} - 1\right) = \frac{3}{2}n - 2 \diamond$$

$$n = 2k + 1 \quad \text{اگر} \diamond$$

$k$  عدد زوج دو تایی  $\diamond$

$Max$  عدد مقایسه برای  $k$  عدد  $Min$  و  $k$  عدد مقایسه برای  $k$  عدد  $\diamond$

به علت وجود یک عنصر اضافی باقی مانده،  $k$  مقایسه برای یافتن عنصر کمینه و  $k$  مقایسه برای عنصر بیشینه  $\diamond$

پس  $3k = \left\lceil \frac{3}{2}n \right\rceil$  مقایسه لازم است :  $\diamond$

$$\left\lceil \frac{3(2k+1)}{2} \right\rceil - 2 = 3k + \left\lceil \frac{3}{2} \right\rceil - 2 = 3k \diamond$$

## راه بهینه ۲

❖ تمرین: یک راه بهینه را به صورت بازگشتی بنویسید (به زبان پایتون) و رابطه زیر را با استفاده از جایگزینی و تکرار حل کنید

❖ راهنمایی:

❖ تقسیم آرایه به ۲ و  $n-2$  عنصر

$$T(n) = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ T(n - 2) + 3 & n > 2 \end{cases}$$

### ۳- قضیه اصلی

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$       ( $a \geq 1, b > 1$ )

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \log n) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ AND } af(n/b) < cf(n) \text{ for large } n \end{cases}$$

$\varepsilon > 0$   
 $c < 1$

شروط استفاده از قضیه اصلی:

۱-  $a \geq 1, b > 1$

۲-  $a$  و  $b$  باید ثابت باشند

۳- تابع  $f(n)$  صعودی باشد

۴- حتما در یکی از این سه حالت روبرو بتوان آن را قرار داد

## مثال - مرتب سازی ادغامی

---

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

اگر با قضیه اصلی بخواهیم به این مساله نگاه کنیم خواهیم داشت:

$$n^{\log_b^a} = n^{\log_2^2} = n$$

$$f(n) \in \Theta(n)$$

و در نتیجه:

$$n^{\log_b^a} \in \Theta(f(n))$$

لذا طبق قضیه اصلی داریم:

$$T(n) \in \Theta(n^{\log_b^a} \lg(n)) = \Theta(n \lg(n))$$

## مثال

---

تابع زیر را با استفاده از قضیه اصلی تحلیل کنید:

$$T(n) = 7T(n/3) + O(n)$$

$$0 < \epsilon < \log_3^7 - 1 \quad \text{برای همه مقادیر } n^{\log_b^a - \epsilon} = n^{\log_3^7 - \epsilon} > n$$

$$T(n) \in \Theta(n^{\log_b^a}) = \Theta(n^{\log_3^7}) \approx \Theta(n^{1.77})$$

## مثال

مثال: تابع زیر را با استفاده از قضیه اصلی تحلیل کنید:

$$T(n) = 8T\left(\frac{n}{2}\right) + 25 \times n^3$$

$$n^{\log_b^a} = n^{\log_2^8} \in \Theta(25 \times n^3)$$

لذا طبق قضیه اصلی خواهیم داشت:

$$T(n) \in \Theta(n^{\log_b^a} \lg(n)) = \Theta(n^{\log_2^8} \lg(n)) = \Theta(n^3 \lg(n))$$

## مثال

مثال: تابع زیر را با استفاده از قضیه‌ی اصلی تحلیل کنید:

$$T(n) = 4T(n/2) + O(n^2 \lg(n))$$

با وجود آن ( $n^2 \log n \in \Omega(n^{\log_b^a} = n^2)$  و لی هیچ مقدار  $\epsilon > 0$  وجود ندارد که به ازای آن ( $n^2 \log n \in \Omega(n^{2+\epsilon})$  چرا که هرچقدر هم مقدار کمتری برای  $\epsilon$  انتخاب کنیم باز هم مقداری از  $n_0$  وجود خواهد داشت که برای  $n > n_0$  داشته باشیم  $\lg(n) > n^\epsilon$ . به همین دلیل هیچ یک از دو تابع  $n^2$  و  $n^2 \lg(n)$  به طور چند جمله‌ای بیشتر از تابع دیگر نیست، و این دو تابع روند رشد یکسانی هم ندارند، لذا نمی‌توانیم از قضیه‌ی اصلی استفاده کنیم.

۲- مروری عرس و قسم اسید که رابطه ما درینی از ساخت تقریبی شده برای تقریبی احصی برآورد کرد. مسئله:

$$T(n) = 2 T\left(\frac{n}{2}\right) + \underbrace{n \log n}_{f(n)}$$

$\left\{ \begin{array}{l} n \log n \in O(n^{1-\epsilon}) \\ n \log n \in \Theta(n) \end{array} \right.$  در حالت

برای مروری این رابطه بگیر (برای  $\epsilon > 0$ )  $\log n \in \Omega(n^\epsilon)$  باید برای همان از تقریب  
اگر و تعداد همواری استفاده کنیم:

$$\textcircled{1} \quad f(n) \in \Omega(g(n)) \xrightarrow[n \rightarrow \infty]{} \lim \frac{f(n)}{g(n)} \xrightarrow[1]{\substack{\downarrow \\ \downarrow}} \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n) \log n}{n^\epsilon} = ?$$

$$\textcircled{2} \quad \text{برای همواری: } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

$$\textcircled{1}, \textcircled{2} \Rightarrow \lim_{n \rightarrow \infty} \frac{\log n}{n^\epsilon} \underset{\substack{\text{مشتق} \\ \text{مشتق}}}{=} \frac{\frac{1}{n \ln 2}}{\epsilon n^{\epsilon-1}} = \frac{\frac{1}{\ln 2}}{\frac{\epsilon}{n^{1-\epsilon}}} = \frac{n^{1-\epsilon} \times \ln 2}{\epsilon} = 0$$

~~پس~~  $\log n \in O(n^\epsilon)$  و  $\log n \in \Omega(n^\epsilon)$  از  $\log n \in \Theta(n^\epsilon)$

---

نشان دادن با قاعده  
هوپیتال و حد در  
او مگا

### ۳-تغییر متغیر

❖ بسیار پیش می‌آید که حل یک رابطه‌ی بازگشتی با تعریف متغیرهای جدید و نوشتن تابع بر اساس آنها آسان‌تر شود.

$$T(n) = 2T(\sqrt{n}) + \lg n$$

$$n = 2^m, m = \log n$$

$$T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + m$$

$$S(m) = T(2^m)$$

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

$$S(m) = \theta(m \log m)$$

$$T(n) \in \theta(m \log m) = \theta(\log n \log \log n)$$

## ۴- حدس و استقرای ریاضی INDUCTION METHOD

---

برنامه بازگشتی ماهیت استقرایی دارند. برای تحلیل اینگونه برنامه‌ها به شیوه استقرایی عمل می‌کنیم:

- ❖ ابتدا الگوریتم بازگشتی را برای آن طراحی کرده و رابطه بازگشتی را می‌نویسیم
- ❖ ۱- پایه: مسئله را به ازای کوچکترین مقدار ورودی  $n_0 = n$  تحقیق می‌کنیم
- ❖ ۲- فرض: فرض می‌کنیم مسئله به ازای  $n < k$  برقرار است
- ❖ ۳- حکم: با استفاده از فرض استقرای و مجموعه فرضیات بدیهی، حکم را ثابت کنیم ( $k=n$ )

# حدس و استقرای ریاضی...

---

در این روش ابتدا سعی می‌کنیم جواب را حدس بزنیم و در مرحله بعد آن را اثبات می‌کنیم.

- اگر بخواهیم اثبات کنیم که  $T(n) = O(f(n))$  است، باید ثابت کنیم که  $T(n) \leq cf(n)$  است. یعنی در مرحله استقرا نشان دهیم که یک مقدار ثابت  $c$  وجود دارد که در این نامعادله صدق می‌کند.
- اگر بخواهیم اثبات کنیم که  $T(n) = \Omega(f(n))$  است باید با استقرا ثابت کنیم که  $T(n) \geq cf(n)$  است.
- اگر بخواهیم اثبات کنیم که  $T(n) = \Theta(f(n))$  است، باید با استقرا ثابت کنیم که اولاً  $T(n) = O(f(n))$  و به علاوه  $T(n) = \Omega(f(n))$  است.

# مثال-الگوریتم مرتب سازی ادغامی

❖ شکستن برنامه به دو زیر آرایه نصف شده و درنهایت مرتب سازی نتایج

❖  $T(n) = 2T(n/2) + n$

❖ حدس:  $O(n \log n)$

❖ ۱- پایه استقرا:  $\forall n = 2 \rightarrow T(2) \leq 2c \lg 2 \rightarrow T(2) \leq 2c \rightarrow c \geq T(2)/2 > 0$

چون  $T(2)$  یک مقدار ثابت مثبت است،  $c \geq T(2)/2$  همیشه برقرار است

❖ ۲- فرض: برای  $k < n$  فرض میکنیم که  $T(k) \leq ck \log k$

❖ ۳- حکم: باید ثابت شود که برای یک  $c > 0$ ، رابطه  $T(n) \leq cn \log n$  برقرار است

## مثال-الگوریتم مرتب سازی ادغامی...

---

- ❖  $T(n) \leq cn\log n$
- ❖  $2T\left(\frac{n}{2}\right) + n \leq cn\log n$
- ❖  $2\left(\frac{cn}{2}\log\left(\frac{n}{2}\right)\right) + n \leq cn\log n$
- ❖  $cn(\log n - 1) + n \leq cn\log n$
- ❖  $n(1 - c) \leq 0$
- ❖  $c \geq 1$

# مثال - جستجوی دودویی

فرض کنید یک آرایه‌ی مرتب با  $n$  عنصر به شما داده شده است و بخواهیم بدانیم آیا عدد  $x$  در این آرایه ظاهر شده است؟

```
def recursive_binary_search(array, x, down, up):
    if down == up -1 :
        if array[down] == x:
            return "FOUND"
        else:
            return "NOT FOUND"
    mid = (up + down) // 2
    if array[mid] == x:
        return "FOUND"
    elif array[mid] < x:
        return recursive_binary_search(array, x, mid, up)
    else:
        return recursive_binary_search(array, x, down, mid)
```

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

## مثال - جستجوی دودویی

حدس:  $O(\log n)$

۱- پایه استقرا:  $n_0 = 2, T(2) \leq c \log 2, c \geq T(2) > 0$  همیشه برقرار است  
چون  $T(2)$  یک مقدار ثابت مثبت است،  $c \geq T(2)$  همیشه برقرار است

۲- فرض: برای  $k < n$  فرض میکنیم که  $T(k) \leq c \log k$

۳- حکم: باید ثابت شود که برای  $n > 0$  یک  $c$  رابطه  $T(n) \leq c \log n$  برقرار است

## مثال - جستجوی دودویی

---

- ❖  $T(n) \leq c \log n$
- ❖  $T\left(\frac{n}{2}\right) + 1 \leq c \log n$
- ❖  $\left(c \log\left(\frac{n}{2}\right)\right) + 1 \leq c \log n$
- ❖  $c(\log n - 1) + 1 \leq c \log n$
- ❖  $(1 - c) \leq 0$
- ❖  $c \geq 1$

## چگونه می توان حدس خوبی زد

---

- ❖ متأسفانه روش کلی و مشخصی برای حدس پاسخ یک رابطه بازگشتی وجود ندارد .
- ❖ حدس زدن درست یک پاسخ نیازمند تجربه و گاهاً ابتکاری است

## ۵- روش درخت بازگشت

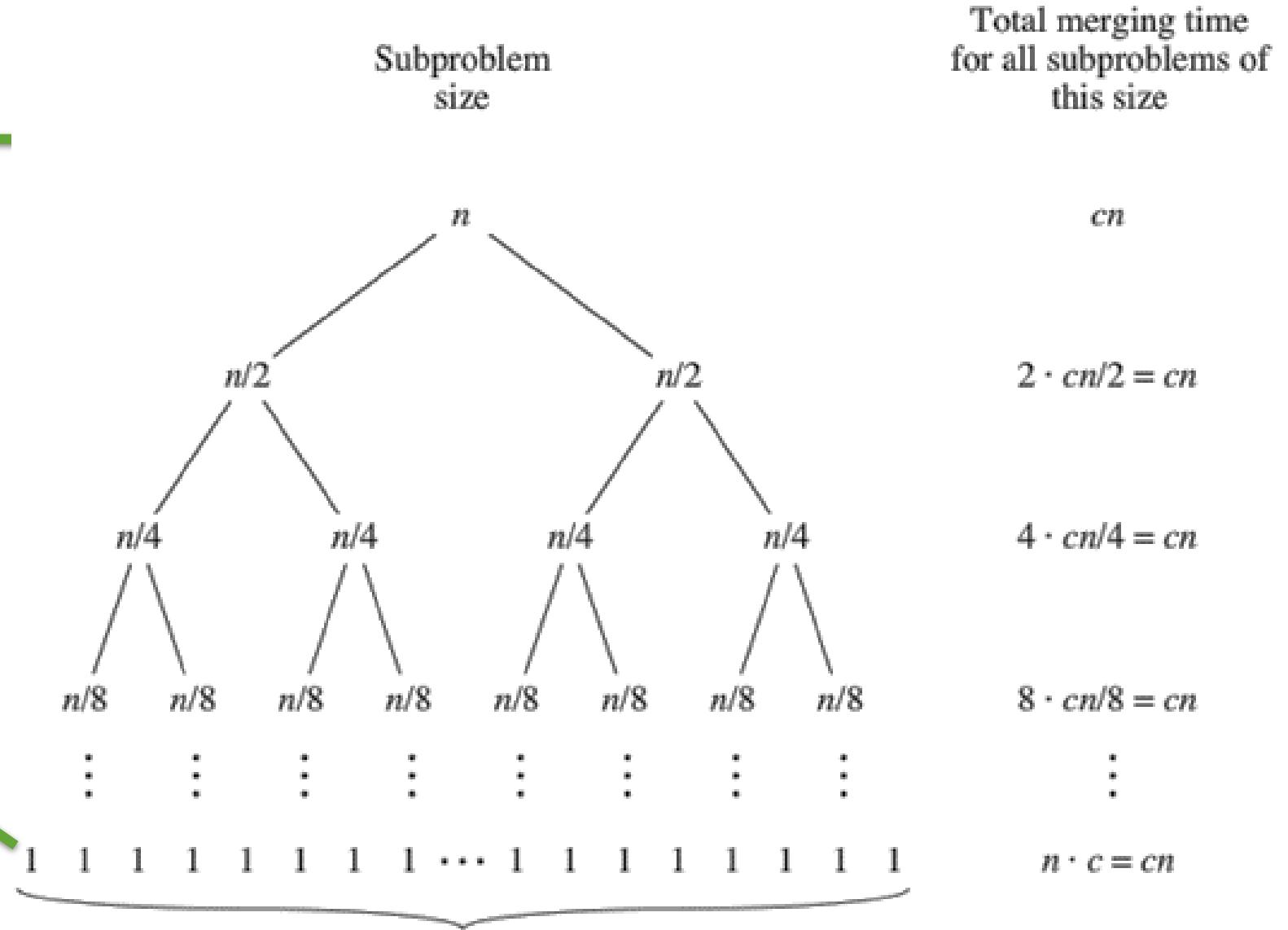
---

رسم یک درخت که روند تابع بازگشتی را در هر سطح نشان دهد

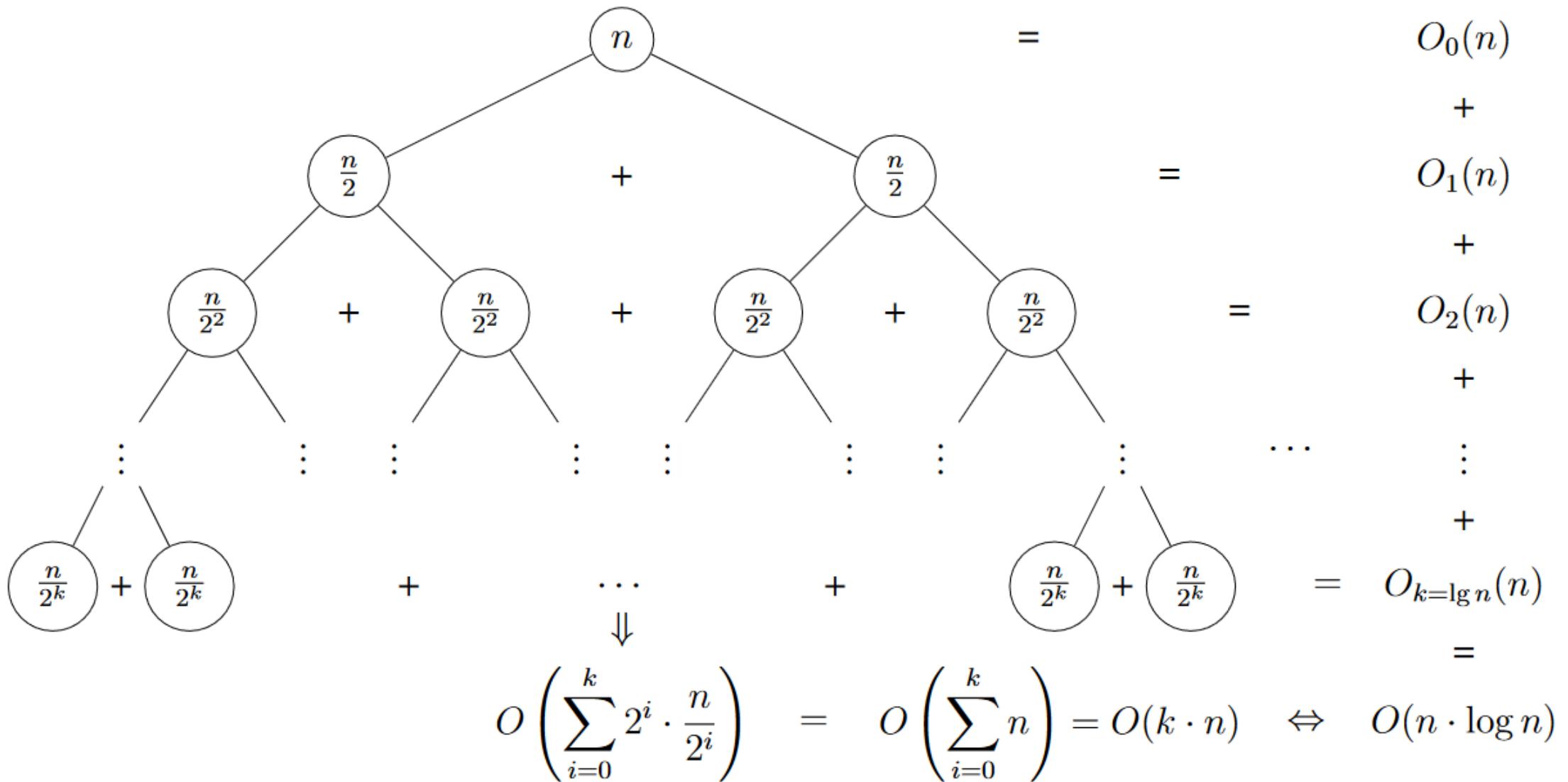
# مثال - مرتب سازی ادغامی

$$T(n) = 2T(n/2) + n$$

$\frac{n}{2^j} = 1 \rightarrow j = \log_2^n$



$$T(n) = 2(n/2) + O(n)$$



## مثال - مرتب سازی ادغامی

در فراخوانی  $\text{ام}$ ، اندازه هر زیر مسئله برابر با  $n/2^j$  و تعداد زیر مسئله ها برابر  $2^j$  می شود.

پس کل هزینه اجرا در مرحله  $\text{ام}$  برابر:

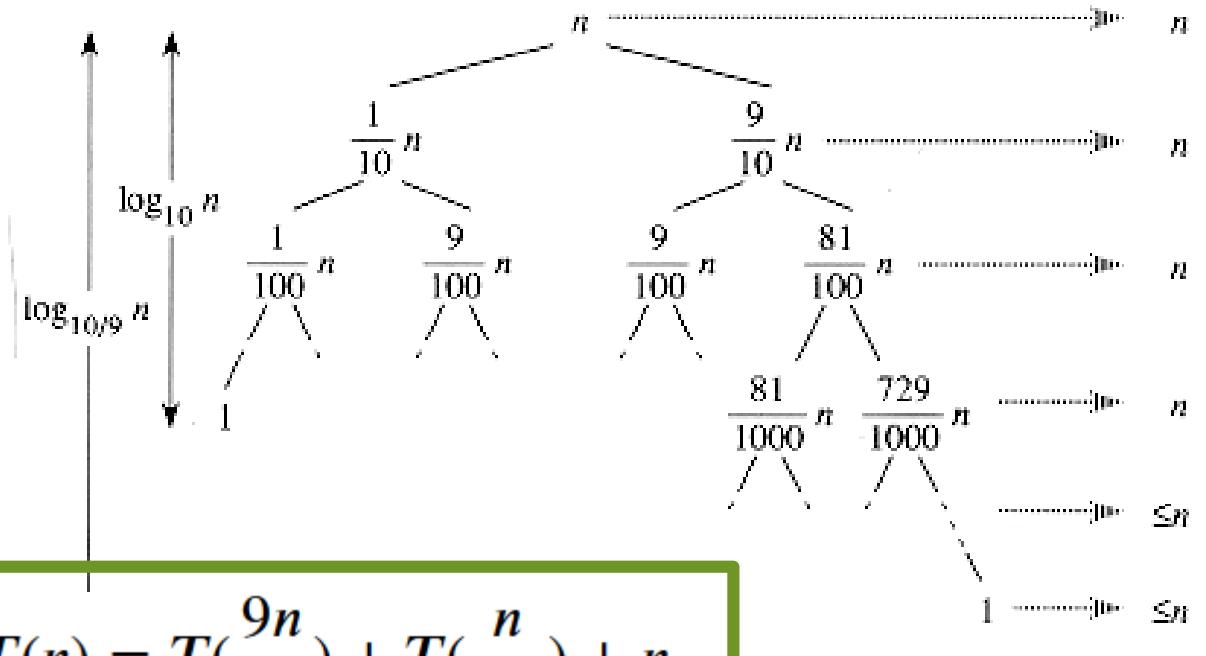
$$\frac{2^j cn}{2^j} = cn$$

از طرفی تعداد مراحل الگوریتم، یک واحد بیشتر از ارتفاع درخت بازگشتی است. برای ساده تر شدن، فرض می شود که  $n$  توانی از دو باشد ( $n = 2^h$ )، که  $h$  ارتفاع درخت است

پس تعداد سطوح درخت:  $h + 1 = 1 + \log n$

هزینه کل:  $cn(1 + \log n) = cn\log n + cn = \theta(n\log n)$

# مثال



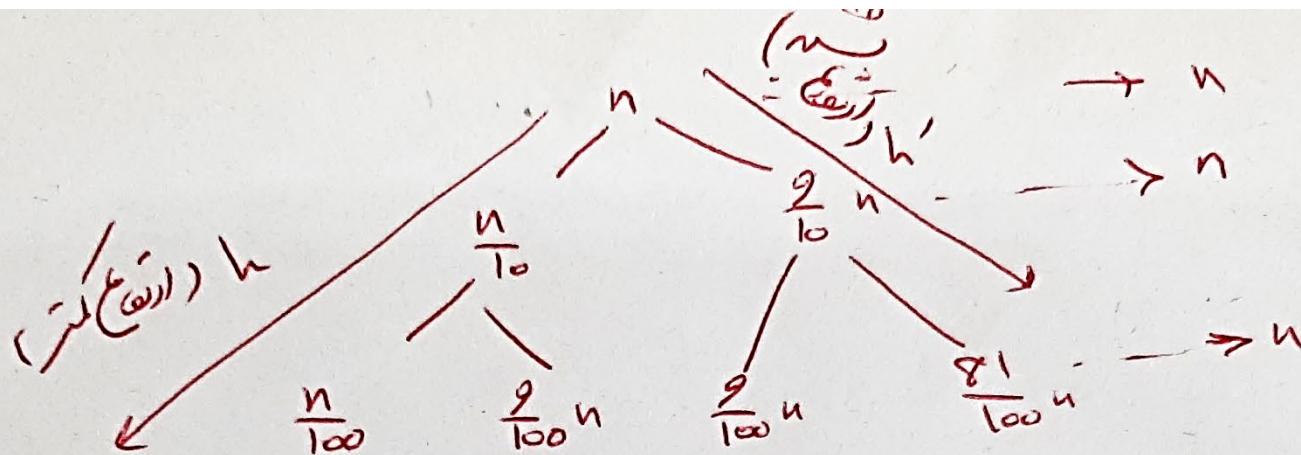
$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + n$$
$$T(1) = 1$$

این درخت نامتوازن است  
و عمق برگ‌های آن بین  $\log_{10/9} n$  تا  $\log_{10} n$  متغیر است.

لذا هزینه‌ی کل این تابع بین  $n \log_{10/9} n$  و  $n \log_{10} n$  متغیر  
است

$$\theta(n \log n)$$

$$\Theta(n \lg n)$$



$$\rightarrow \frac{n}{10^h} = 1 \rightarrow h = \log_{10} n$$

$$\rightarrow \left(\frac{9}{10}\right)^{h'} = 1 \rightarrow h' = \log_{\frac{10}{9}} n$$

حصصي  $\log_{\frac{10}{9}}$   
أكبر من  $\log_{10}$

$$T(n) \in O(n \log_{\frac{10}{9}} n) \quad \text{و} \quad T(n) \in \Omega(n \log_{10} n)$$

$$\log_B(N) = \frac{\log_2(N)}{\log_2(B)} = k \times \log_2(N)$$

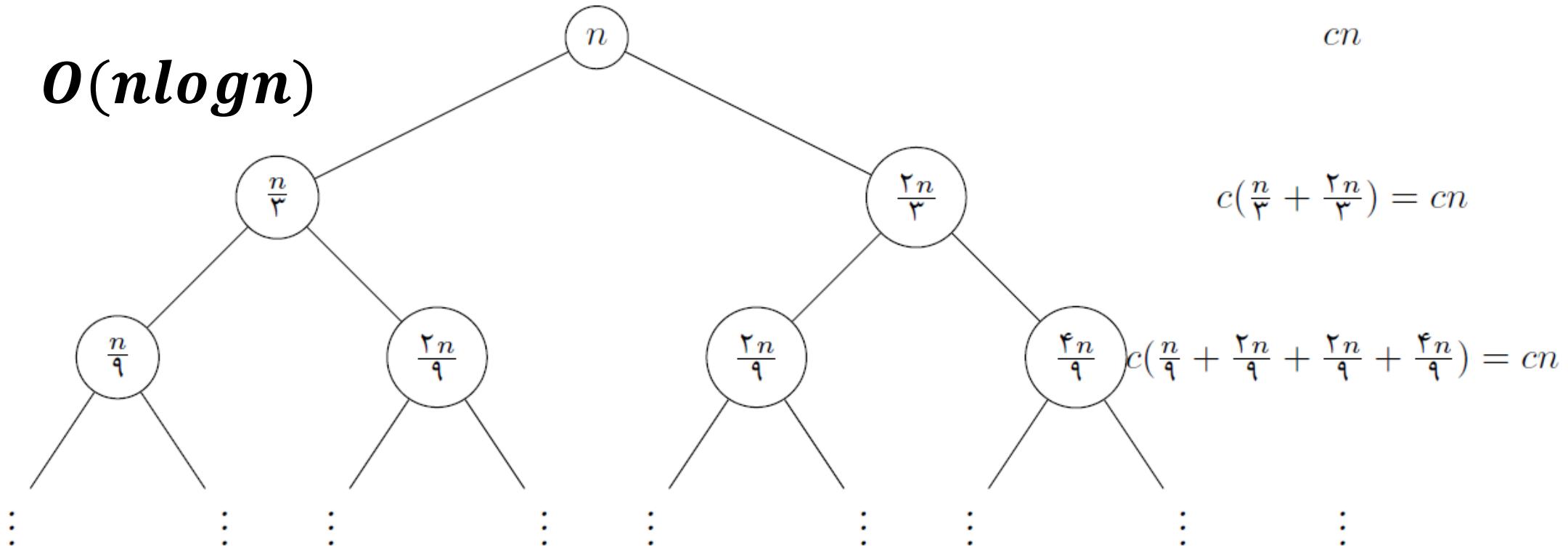
ممكن أن  $k$  يختلف، بحسب الاعداد المدخلة في خوارزمية الـ  $\log$

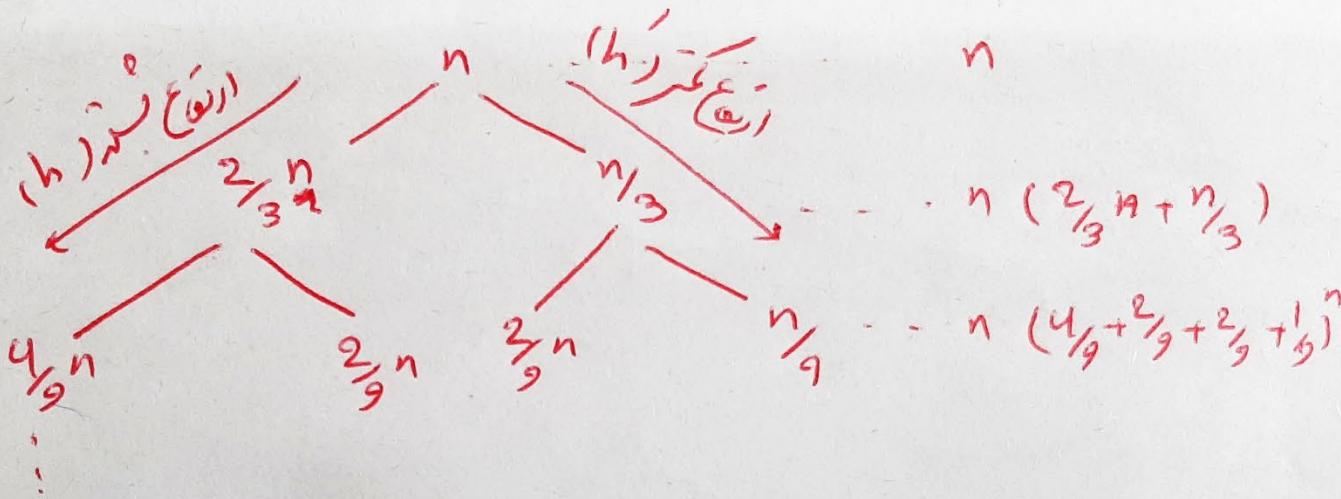
$$T(n) \in \Theta(n \log n)$$

# مثال

$$T\left(\frac{n}{r}\right) + T\left(\frac{rn}{r}\right) + \Theta(n)$$

**$O(n \log n)$**

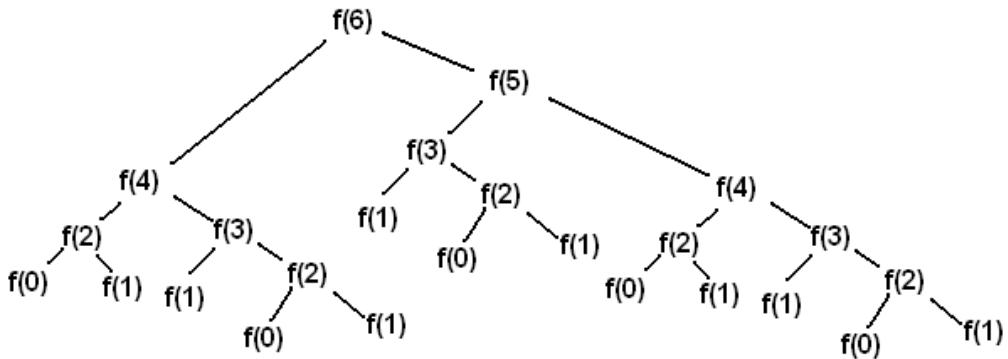




$$\Rightarrow \left(\frac{2}{3}\right)^h n < 1 \rightarrow h = \log_{\frac{3}{2}}^n \Rightarrow T(n) \in \Theta(n \log n)$$

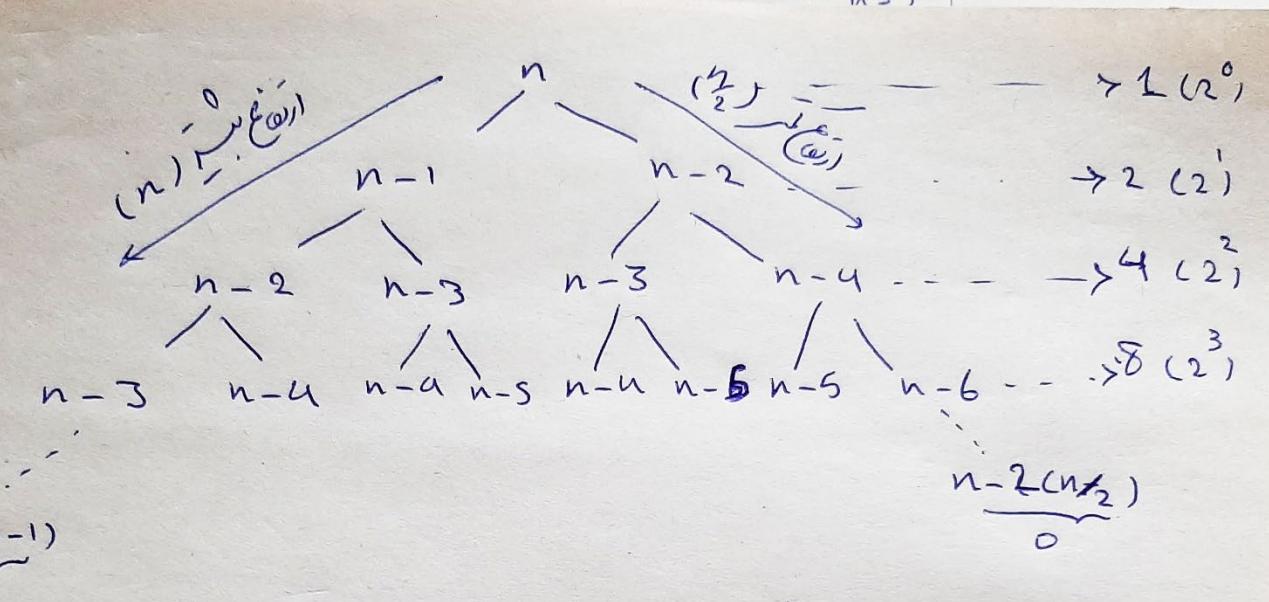
$$\Rightarrow \left(\frac{1}{3}\right)^{h'} n = 1 \Rightarrow h' = \log_3^n$$

# مثال - فیبوناچی



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

$T(n) = \Omega(2^{n/2})$  (lower bound)  
 $T(n) = O(2^n)$  (upper bound)  
 $T(n) = \Theta(\text{Fib}(n))$  (tight bound)



$$1 + 2 + 2^2 + \dots + 2^{n-1} = 2^n - 1 \Rightarrow T(n) \in O(2^n)$$

دیگر تقریب (ترکیبی):  $86,00 \Leftarrow$

$$1 + 2 + 2^2 + \dots + 2^{n_2} = 2^{n_2+1} - 1 \Rightarrow T(n) \in \Omega(2^{n_2})$$

دیگر تقریب (ترکیبی):  $86,00 \Leftarrow$

# مقایسه پیچیدگی زمانی الگوریتم های مرتب سازی

SORTING ALGORITHM	TIME COMPLEXITY		
	BEST CASE	AVERAGE CASE	WORST CASE
Bubble Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	$O(N^2)$
Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Merge Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$

# مرتب سازی شمارشی

```
#counting sort for positive numbers
A=[1,2,3,1,4,5,1,6,7,1,8,9,2,1]
n=len(A)
m=max(A)+1
count=[0]*m
for i in A:
    count[i]+=1
print(count)
A=[]
for i in range(m):
    A+=[i]*count[i]
print(A)
```

---

function COUNTINGSORT( $A[1\dots n]$ ,  $k$ )

for  $j = 0$  to  $k$  do

$C[j] \leftarrow 0$

for  $j = 1$  to  $n$  do

$C[A[j]] \leftarrow C[A[j]] + 1$

$count \leftarrow 1$   $\theta(n + k)$

for  $j = 0$  to  $k$  do

for  $i = 1$  to  $C[j]$  do

$B[count] \leftarrow j$

$count \leftarrow count + 1$

return  $B$

# نکات الگوریتم مرتب سازی شمارشی

- ❖ مرتب سازی پایدار
- ❖ در صورتی که  $k$  خیلی بزرگتر از  $n$  باشد، این روش بهینه نیست.
- ❖ اگر  $k = O(n)$  باشد، این الگوریتم از مرتبه  $O(n)$  است و یک الگوریتم خطی است (بهینه ترین روش مرتب سازی)
- ❖ با توجه به این که تعداد وقوع هر عنصر آرایه  $A$  در `count` نگهداری می‌شود، عملاً استفاده از این روش برای بازه‌های بزرگ اعداد غیرعملی است. (مناسب برای مرتب سازی اعداد ۸ بیتی ( $2^8 = k$ ) و نامناسب برای ۳۲ بیتی ( $2^{32} = k$ ))

تحلیل دقیق تر در حالت متوسط :

$$O\left(k * \left(\frac{n}{k} + c\right)^2\right) = O\left(\frac{n^2}{k} + c' k\right) \\ = O(n + k)$$

# مرتب سازی سطلی

```
#Bucket sort
def Bucket_sort(A):
    bins=[[] for i in range(10)] #define
    n=len(A)
    for i in A:
        bins[i//10]+=[i]
    A=[]
    for i in range(10):
        A+=Insert_sort(bins[i])
    print(A)
return(A)

A=[21, 32, 1, 4, 5, 0, 66, 40, 99, 85]
A = Bucket_sort(A)
print(A)
```

$O(n)$

تعداد باكت ها: K:

بدترین حالت:  
همه اعداد داخل یک سطل: ( $O(n^2)$ )

حالات متوسط (توزیع نرمال عناصر ورودی):  
 $O(k * (\text{متوسط عناصر موجود در هر باكت})) =$

$$O\left(k * \left(\frac{n}{k}\right)^2\right) = O\left(\frac{n^2}{k}\right), k \in \Theta(n) \\ = O(n)$$

## مرتب سازی مبنا یی

```
A=[163, 456, 782, 345, 9856, 123, 987, 345]
radix=1

while radix<=max(A):
    bins=[[0]*10 for _ in range(10)]
    for i in A:
        bins[(i//radix)%10]+=[i]
    radix*=10

A=[]
for i in range(10):
    A+=bins[i]

print(A)
```

$$\theta(d(n+b)) = \theta(nd)$$

# حداکثر تعداد ارقام در عدد: d: مینا: b:

## مرتب سازی مبنایی

---

- ❖ اعداد  $d$  رقمی و مرتب سازی بر اساس کم ارزش ترین رقم مرتب می شوند و سپس دومین رقم و ...
- ❖ مرتب سازی پایدار و بسیار پرکاربرد در تهیه واژه نامه ها و مرتب سازی اعداد

# مقایسه پیچیدگی زمانی الگوریتم های مرتب سازی

	SORTING ALGORITHM	TIME COMPLEXITY		
		BEST CASE	AVERAGE CASE	WORST CASE
با مقایسه Comparison sorts	Bubble Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
	Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	$O(N^2)$
	Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
	Merge Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
بدون مقایسه Non-comparison sorts	Radix Sort	$\Omega(N k)$	$\Theta(N k)$	$O(N k)$
	Count Sort	$\Omega(N + k)$	$\Theta(N + k)$	$O(N + k)$
	Bucket Sort	$\Omega(N + k)$	$\Theta(N + k)$	$O(N^2)$