طراحی و تحلیل الگوریتم ها

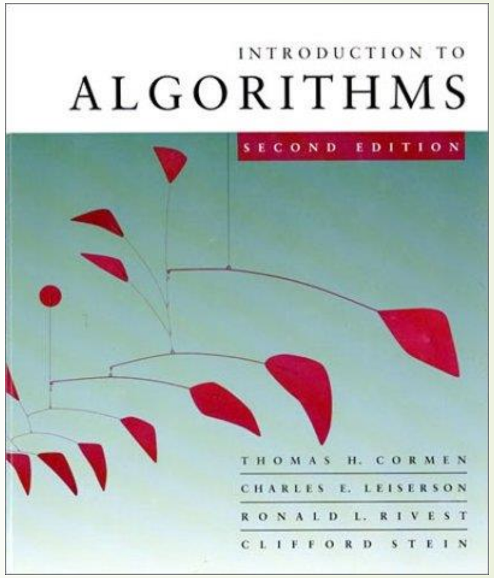
دکتر امیر لکی زاده

استادیار گروه مهندسی کامپیوتر دانشگاه قم

منابع

1. Introduction to Algorithm(Third Edition)

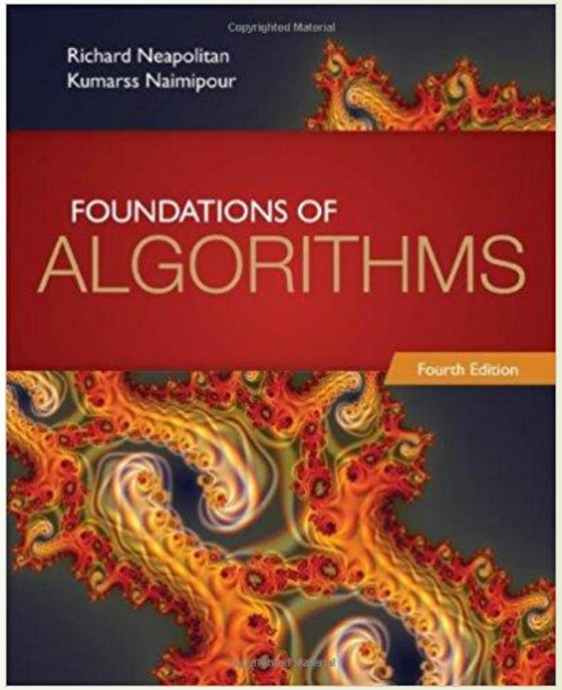
- Thomas H.**C**ormen
- Charles E.**L**eiserson
- Ronald L.**R**ivest
- Clifford **S**tein



منابع

2. **Foundations Of Algorithms**(Fourth edition)

- Richard Neapolitan
- Kumarss Naimipour



مقدمه

- **Algorithm:**
- An Algorithm is any well – defined computational Procedure that takes some value, or set of values as inputs and Produces some value, or set of values as output.
- یک الگوریتم، یک رویه محاسباتی خوش تعریف است که یک مقدار یا مجموعه ای از مقادیر را به عنوان ورودی می گیرد و یک مقدار یا مجموعه ای از مقادیر را به عنوان خروجی تولید می کند.
- An Algorithm as a tools for salving a well – specified computational problem.
- یک الگوریتم، ابزاری برای حل کردن یک مسأله محاسباتی است.

مقدمه

- **Sorting problem:**
- Input: A Sequence of n numbers $<a, a_2 \ldots a_n>$
- Output:A permutation$<a'_1, a'_2, \ldots a'_n> \ of\ input\ sequence\ such\ that\ a'_1 < a'_2 < \cdots < a'_n$
- فرض کنید دنباله <10, 7, 1, 2> یک نمونه (Instance) از مسأله مرتب سازی (sorting problem) می باشد.   خروجی ←<1, 2, 7, 10>
- **تعریف: الگوریتم صحیح correct Algorithm**
- An algorithm is said to be correct if for every input instance it halts with the correct output.

اگر الگوریتم برای هر نمونه از ورودی، خروجی درستی را تولید کند و متوقف شود می گوییم الگوریتم صحیح است.

مقدمه

- 1. How to design An Algorithm?
- 2. How to Analysis An Algorithm?

مقدمه

- **انواع مسائل قابل حل بوسیله الگوریتم ها:**
- در شبکه اینترنت (ارتباطات): مسیریابی، احراز هویت، پروتکل های ارسال و دریافت، و ...
- امنیت: سیستم های پیشگیری از نفوذ(IPS)، تشخیص نفوذ(IDS)، تشخیص ناهنجاری و ...
- رمزنگاری(Cryptography)، متقارن ـ نامتقارن، PKI و ...
- تجارت الکترونیکی: رمزنگاری، امضای دیجیتالی، ذخیره سازی داده ها.
- اختصاص منابع کمیاب برای بدست آوردن بیشترین شود. مثال: حداکثر استفاده از cpu و حافظه.
- محاسبات علمی(ریاضی، آمار، زمین شناسی، هواشناسی و ...)، محاسبات فنی
- بیوانفورماتیک، زیست شناسی محاسباتی و زیست سامانه ها
- تحلیل شبکه های اجتماعی
- داده کاوی و متن کاوی
- بازیابی اطلاعات
- محاسبات توزیع شده

مقدمه

- روش های ضرب n ماتریس (پرانتزگذاری)

$A_1 \rightarrow (A_1)$

$A_1A_2 \rightarrow (A_1.A_2)$

$A_1A_2A_3 \rightarrow ((A_1.A_2).A_3),(A_1.(A_2.A_3))$

$A_1A_2A_3A_4 \rightarrow ((A_1.A_2).(A_3.A_4)),(((A_1.A_2).A_3).A_4),$
$((A_1.(A_2.A_3)).A_4),(A_1.((A_2.A_3).A_4)),$
$(A_1.(A_2.(A_3.A_4)))$

مقدمه

- **تعریف الگوریتم کارا: efficient Algorithm**
- یک الگوریتم را کارا گویند اگر در زمان چند جمله ای برحسب اندازه ورودی تصمیم پذیر قطعی باشد. (مسئله را حل کنید)
- مسائل را از لحاظ زمان لازم برای حل مسأله (زمان به عنوان تابعی از اندازه ورودی مسأله) به دو دسته تقسیم می شوند:
- ۱. مسائل تصمیم پذیر قطعی در زمان چند جمله ای:

Deterministic decidable in polynomial time(P)

- ۲. مسائل تصمیم پذیر غیر قطعی در زمان چند جمله ای:

NonDeterministic decidable in polynomial Time(NP)

مقدمه

منظور از مسایل تصمیم پذیر قطعی در زمان چند جمله ای(کلاس P):

برای حل آن یک الگوریتم کارا وجود دارد مانند مسئله مرتب سازی یا

الگوریتمی برای آن مسأله وجود دارد که در آن تعداد عملیات برای حل مسأله به صورت یک تابع چند جمله ای از اندازه ورودی مسأله می باشد.

```
int test (int n)
{
int i = 0 int s = 0 ;
for (i = 1; i <= c; i ++)
s = s + 1
return s;
}
```

$T(n) = 4 + (C + 1) + 2C = 5 + 3C$

- تعداد: محاسبات الگوریتم T (n)، مستقل از اندازه ورودی می باشد و همواره مقدار ثابتی است.

مقدمه

- **مسائل تصمیم پذیر غیر قطعی در زمان چند جمله ای(کلاس NP)**

الگوریتمی در زمان چند جمله ای وجود دارد که به ازای یک نمونه از آنها تصمیم پذیر است.

مسائل NP_C:

مسائلی هستند که هیچ الگوریتم تصمیم پذیر در زمان چند جمله ای تاکنون برای آنها ارائه نشده است و از طرفی ثابت نیز نشده است که چنین الگوریتمی برای آنها وجود ندارد.

- No efficient Algorithm for an NP – Complete Problem has been found and nobody has ever proven that efficient Algorithm for it can not exist
- مثال: فروشنده دوره گرد Traveling Salesman Problem(TSP)

مقدمه

- توجه: تعریف مسائل NP – Complete (NP_C)
- یک مسأله را NP_C گو یند اگر:

۱- NP باشد

۲ـ تمامی مسائل NP در زمان چند جمله ای قابل تبدیل به آن باشند: reducible in polynomial time

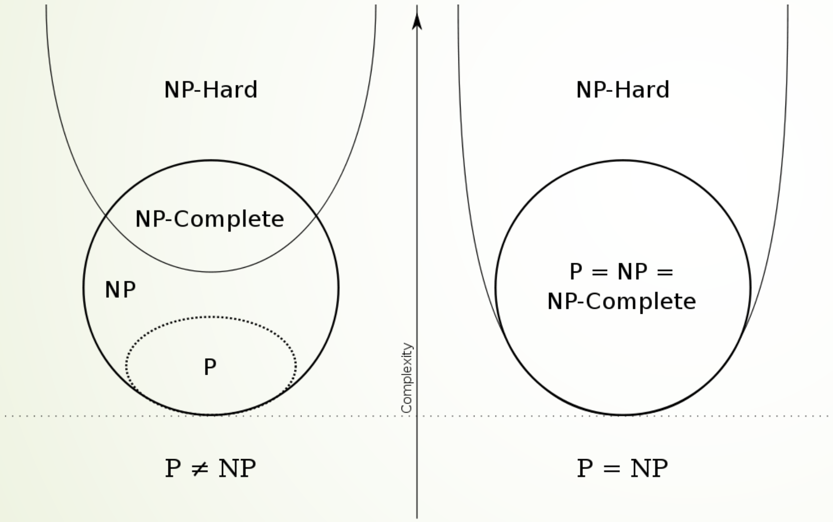
مقدمه

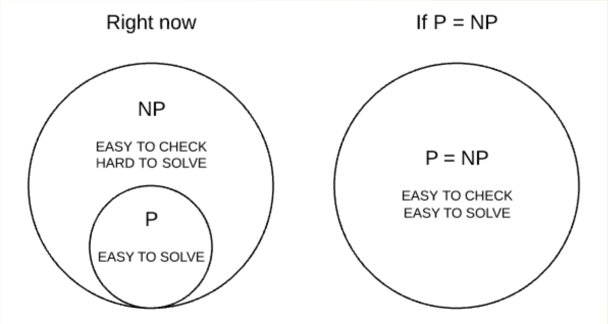
- چرا مسائل NP_C مورد علاقه هستند؟

۱- اگر یک الگوریتم تصمیم پذیر در زمان چند جمله ای برای یک مسأله NP_C یافت شود آنگاه راه حل فوق قابل تعمیم برای تمام مسائل NP خواهد بود و در نتیجه P = NP می باشد.

۲- اگر ثابت کنیم یک مسأله NP_C است می توانیم به جای اینکه وقت خود را صرف پیدا کردن یک الگوریتم بهینه به هدر بدهیم سعی کنیم یک الگوریتم را که یک جواب خوب و نزدیک به بهینه به دست می دهد، ارائه کنیم.

مقدمه


NP-Hard
NP-Complete
NP
P
P ≠ NP
Complexity
NP-Hard
P = NP =
NP-Complete
P = NP


Right now
NP
EASY TO CHECK
HARD TO SOLVE
P
EASY TO SOLVE
If P = NP
P = NP
EASY TO CHECK
EASY TO SOLVE

تحلیل الگوریتم ها

- مقایسه کارایی (efficiently) دو الگوریتم
- **Faster Computer or Faster Algorithms**

Problem: Sorting,

Alg1: Insertion sort T(n)= $2n^2$

Alg2: Merge sort T(n)= $50n.\log n$

Suppose have two computers:

A: 10^9 Ins/sec , B: 10^7 Ins/sec,   A is 100’ faster

$$\frac{2 \cdot (10^6)^2 \text{ instructions}}{10^9 \text{ instructions/second}} = 2000 \text{ seconds},$$

while computer B takes

$$\frac{50 \cdot 10^6 \lg 10^6 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 100 \text{ seconds}.$$

تحلیل الگوریتم ها

- الگوریتم مرتب سازی درجی: (مناسب برای تعداد کمی از عناصر)

$$
\begin{array}{l}
Insertion - Sort\ (A) \\
for\ j \leftarrow 2\ to\ \ length\ [A] \\
\qquad \{Key\ \leftarrow A\ \ [j];\ i \leftarrow j - 1; \\
\qquad while\ ((i > 0) and\ (A[j] > key)) \\
\qquad\qquad \{A\ [i+1] \leftarrow A\ [i]; \\
\qquad\qquad\qquad i - -;\} \\
\qquad A\ [i+1] \leftarrow key; \\
\}
\end{array}
$$

- تعداد عملیات مرتب سازی درجی در بهترین حالت (best case): $n - 1$

تحلیل الگوریتم ها

j	i
2	$1 \to 1 \quad (1)$
3	$2 \to 1 \quad (2)$
4	$3 \to 1 \quad (3)$
. . .	
n	$n-1 \to 1 \ \ (n-1)$

ـ تعداد عملیات مرتب سازی درجی در بدترین حالت (worst case):

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

تحلیل الگوریتم ها

- منظور از تحلیل الگوریتم ها پیش بینی منابع مورد نیاز توسط برنامه می باشد این منابع چند دسته اند.

  ۱ـ زمان اجرا

  ۲ـ حافظه

  ۳ـ سخت افزار

  ۴ـ پهنای باند ارتباطی

- در بحث تحلیل الگوریتم ها به تحلیل زمان اجرای آنها می پردازیم.
- توجه: زمان اجرا running time تابعی از اندازه ورودی می باشد.
- در مسأله مرتب سازی، اندازه ورودی تعداد بیت ها برای نمایش اعداد می باشد.
- زمان اجرای یک الگوریتم بر روی یک ورودی خاص برابر با تعداد عملیات اساسی یا تعداد گام های اولیه که باید اجرا شود.

تحلیل الگوریتم ها

- چرا همواره به دنبال مرتبه زمانی در بدترین حالت هستیم؟

۱ـ بدست آوردن یک کران بالا یا بدست آوردن یک تضمین که الگوریتم بدتر از این کران نمی شود.

۲ـ احتمال رخداد فراوان بدترین حالت

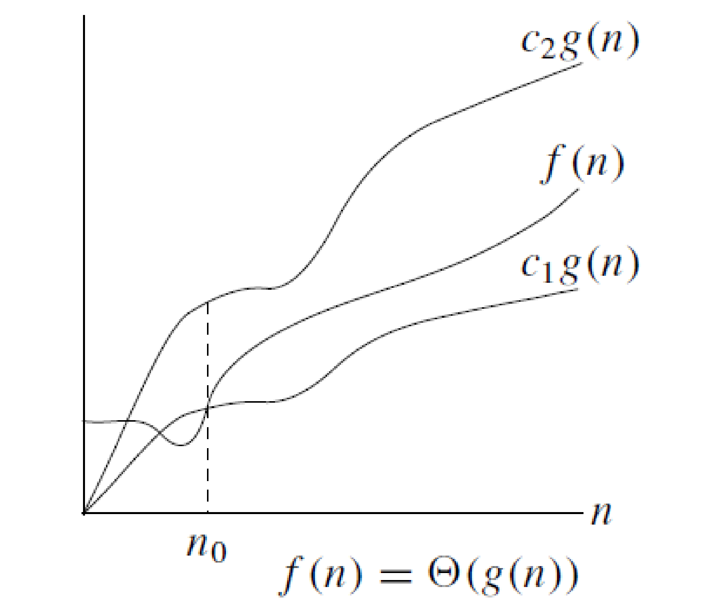
۳ـ در بیشتر مواقع زمان اجرا در حالت میانگین (average case) به اندازه بدترین حالت (worst case) می باشد.

تحلیل الگوریتم ها

- نماد θ

$f, g : N \to N$

- $\theta(g(n)) = [f(n) \quad | \text{there exist positive constants } C_1, C_2, n_0 \text{ such that } \forall n \geq n_o$
  $: C_1 g\,(n) \leq f\,(n) \leq C_2\, g\,(n) \}$



تحلیل الگوریتم ها

- $f\ (n) \in\ \theta\left(g(n)\right) \rightarrow (f(n) = \theta(g(n))$
- به این معنی است که آهنگ افزایش مقادیر دو تابع f(n) و g(n) به ازای مقادیر به اندازه کافی بزرگ $n \geq n_0$ با هم برابر باشند.
- ثابت کنید $f(n) = \theta(g(n))$

$$f\ (n) = \frac{1}{2}n^2 - \ 3\ n \qquad g\ (n) = \ n^2$$

$$C_1n^2 \leq \frac{1}{2}n^2 - 3n \leq C_2n^2$$

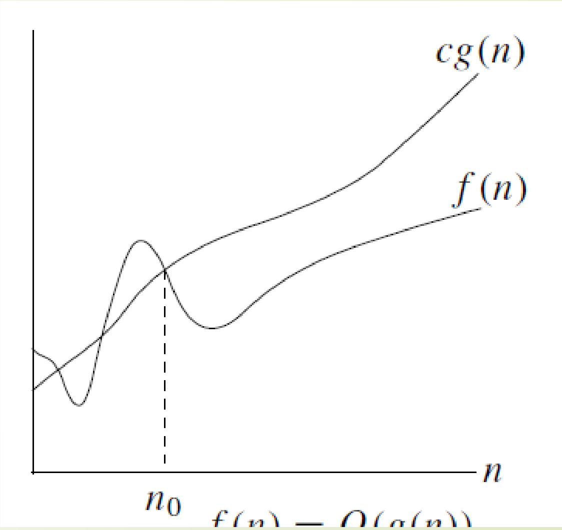
$$C_1n^2 \leq \frac{1}{2}n^2 - 3n \rightarrow C_1 \leq \frac{1}{2} - \frac{3}{2} : n_0 = 7\ , C_1 = \frac{1}{14}$$

$$\frac{1}{2}n^2 - 3n \leq C_2n^2 \rightarrow \frac{1}{2} - \frac{3}{n} \leq C_2 : C_2 = \frac{1}{2}$$

تحلیل الگوریتم ها

- نماد O بزرگ: Big_O ـ

- $O\ (g(n)) = \ \{f\ (n) |\ there\ exist\ Positve\ constants\ C, n_0\ such\ that\ \forall$
  $n \ge n_0 :\ \ f\ (n) \le c\ g(n)\}$



- $f\ (n) \in O\ (g(n)) \rightarrow f(n) = O\ \big(g(n)\big)$
- g(n) یک کران بالا برای f(n) است و یا آهنگ رشد g(n) بیشتر از f(n) می باشد

تحلیل الگوریتم ها

- $f(n) = 100n + 5\, , g(n) = n^2$
- $f(n) = O\ (g(n))$

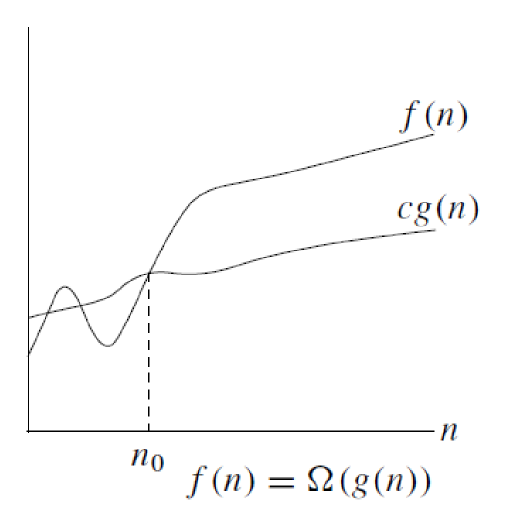
- برای اثبات $Cn^2 \geq 100n + 5$ ارائه یک جفت مقادیر کافی است: $(n_0 = 10^3 , C = 1)$ یا $(n_0 = 1, C = 105)$
- از نماد Big_O برای بیان زمان اجرای الگوریتم در بدترین حالت استفاده می شود.
- اگر T (n) پیچیدگی زمانی یک الگوریتم را نمایش دهد، $T(n) = O(g(n))$ به این معنی است که g(n) کوچکترین کران بالا برای $T(n)$ می باشد یا زمان اجرای الگوریتم در بدترین حالت برابر g(n) می باشد.
- نکته: اگر برای ثابت K داشته باشیم $f(n) = O\ (n)$ گوییم که $f(n)$ محدود چند جمله ای است.

تحلیل الگوریتم ها

- نماد Big – Omega (Ω):

- $\Omega(g(n)) = \{f(n) \quad | ther\ exist\ positvie\ constant\ C_0\ , n_0\ such\ that\ \forall\ n \geq n_0 \qquad f(n) \geq C\ (g(n)\}$

- $f\ (n) \in \Omega(g(n)) \rightarrow f(n) = \Omega(g(n))$



- g(n) یک کران پایین برای f(n) است و یا آهنگ رشد f(n) بیشتر از g(n) می باشد

تحلیل الگوریتم ها

- نماد Big – Omega (Ω):

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

- $g(n)$ یک کران پایین برای $f(n)$ است (آهنگ رشد $g(n)$ کوچکتر از $f(n)$ می باشد).
- از نماد Big_omega برای بیان زمان اجرای الگوریتم در <u>بهترین حالت</u> استفاده می شود.

- اگر T (n) زمان اجرای یک الگوریتم باشد. آن گاه $T(n) = \Omega(g(n))$، به این معنی است که تابع $g(n)$ بزرگترین کران پایین (زمان اجرای الگوریتم در <u>بهترین حالت</u>) می باشد.

تحلیل الگوریتم ها

- نماد o – Small

- $o(g(n)) = \{f(n) \quad | for\ any\ positive\ constant\ C, thete\ exits\ n_0 > 0$
  $such\ that\ \forall\ n \geq n_0\ f(n) < cg(n)\}$
- $f(n) = o(g(n)) \Leftrightarrow f(n) = O(g(n)), f(n) \neq \theta(g(n))$
- $f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$

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

تحلیل الگوریتم ها

- نماد Small – Omega (ω)
- $\omega\left(g(n)\right) = \{f(n) \quad | for\ any\ positive\ constant\ C, thete\ exits\ n_0 > 0$
  $such\ that\ \forall\ n \geq n_0\ f(n) > cg(n)\}$
- $f(n) = \omega\left(g(n)\right) \Leftrightarrow f(n) = \Omega(g(n)), f(n) \neq \theta\left(g\ (n)\right)$
- $f(n) = \omega\left(g(n)\right) \Rightarrow f(n) = \Omega\left(g(n)\right)$

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

تحلیل الگوریتم ها

Transitivity:

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \quad \text{imply} \quad f(n) = \Theta(h(n)),$$
$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \quad \text{imply} \quad f(n) = O(h(n)),$$
$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \quad \text{imply} \quad f(n) = \Omega(h(n)),$$
$$f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \quad \text{imply} \quad f(n) = o(h(n)),$$
$$f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \quad \text{imply} \quad f(n) = \omega(h(n)).$$

تحلیل الگوریتم ها

Reflexivity:

$$
\begin{aligned}
f(n) &= \Theta(f(n)) \,, \\
f(n) &= O(f(n)) \,, \\
f(n) &= \Omega(f(n)) \,.
\end{aligned}
$$

Symmetry:

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)) \,.$$

Transpose symmetry:

$$
\begin{aligned}
f(n) &= O(g(n)) \quad \text{if and only if} \quad g(n) = \Omega(f(n)) \,, \\
f(n) &= o(g(n)) \quad \text{if and only if} \quad g(n) = \omega(f(n)) \,.
\end{aligned}
$$

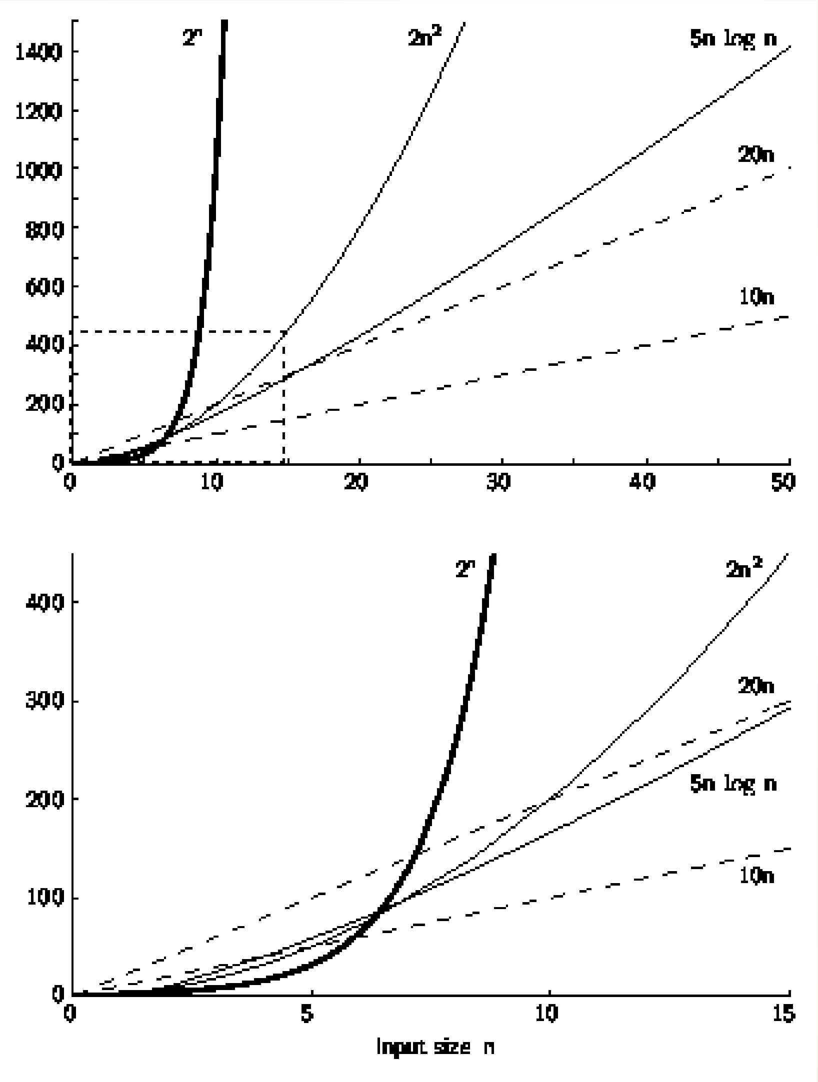
تحلیل الگوریتم ها

$$
\begin{aligned}
f(n) &= O(g(n)) && \approx && a \le b\,, \\
f(n) &= \Omega(g(n)) && \approx && a \ge b\,, \\
f(n) &= \Theta(g(n)) && \approx && a = b\,, \\
f(n) &= o(g(n)) && \approx && a < b\,, \\
f(n) &= \omega(g(n)) && \approx && a > b\,.
\end{aligned}
$$

تحلیل الگوریتم ها

- نمادهای θ,ω,Ω,O,Q نمی توانند روی همه توابع تعریف شوند. (مثل توابع نوسانی)
- $f(n) = n^2 \qquad g(n) = n^{1+sin(n)}$
- اثبات کنید:
- $P\,(n) = \sum_{i=0}^{K} a_i n^i \qquad P(n) = \theta(n^K)$
- $C_1 n^k \le a_0 + a_1 n + \cdots + a_k n^k \le C_2 n^k$

تحلیل الگوریتم ها



تحلیل الگوریتم ها

Example 1.

```
/*return Position of largest value in "A" */
int largest(int[] A, int n) {
  int currlarge = 0;  // Position of largest
  for (int i=1; i<n; i++)
    if (A[currlarge] < A[i])
       currlarge = i; // Remember pos
  return currlarge;   // Return largest pos
}
```

تحلیل الگوریتم ها

Example 2.

```
a = b;
```

This assignment takes constant time, so it is $\Theta(1)$.

Example 3.

```
sum = 0;
for (i=1; i<=n; i++)
   sum += n;
```

تحليل الگوريتم ها

Example 4.

```
sum = 0;
for (i=1; i<=n; i++)
   for (j=1; j<=n; j++)
      sum++;
}
```

i	j	
١	1..n	n بار
٢	1..n	n بار
...		
n	1..n	n بار

تحلیل الگوریتم ها

Example 5.

```
sum2 = 0;
for (j=1; j<=n; j++)
   for (i=1; i<=j; i++)
      sum2++;
```

j	i	
1	1..1	1 بار
2	1..2	2 بار
…		
n-1	1..n-1	n-1 بار
n	1..n	n بار

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

تحلیل الگوریتم ها

Example 6.

```
sum = 0;
for (j=1; j<=n; j++)
   for (i=1; i<=j; i++)
      sum++;
for (k=0; k<n; k++)
   A[k] = k;
```

تحلیل الگوریتم ها

Example 7.

```
sum1 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=n; j++)
    sum1++;
```

تحلیل الگوریتم ها

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.	$\lg(n!)$	$\lg(n^n)$					

تحلیل الگوریتم ها

3-3 Ordering by asymptotic growth rates

a. Rank the following functions by order of growth; that is, find an arrangement $g_1, g_2, \ldots, g_{30}$ of the functions satisfying $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, $\ldots$, $g_{29} = \Omega(g_{30})$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$.

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	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	$n \lg n$	$2^{2^{n+1}}$

تحلیل الگوریتم ها

3-4 ***Asymptotic notation properties***

Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following conjectures.

a. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

b. $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

c. $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$, where $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n.

d. $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.

e. $f(n) = O((f(n))^2)$.

f. $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.

g. $f(n) = \Theta(f(n/2))$.

h. $f(n) + o(f(n)) = \Theta(f(n))$.