Solutions to CLRS

@kevin5396

April 3, 2016

Contents

Part I

Foundations

Chapter 1

The Role of Algorithms in Computing

1.1 Exercises

1.1.1

A sorting example is that after a exam, teachers need sort the students' marks.
I don't know any thing about computing a convex hull xD.

1.1.2

The space that a procedure uses, the resources it consumes.

1.1.3

I've learned Linked List before .

strengths convient to insert and erase.

limitations cost a lot to visit a specific element.

1.1.4

Not so similar. The shortest-path problem is to find the shortest path from a source place to many other places. But the traveling-salesman problems differs, though, is to find a round tour that is the shortest.

1.1.5

Let's take the sorting problem, sorting is exactly needing a best solution. But talking about traveling problem, an approximately algorithm is sufficient.

1.2 Exercises

1.2.1

A topology sort is of great use in find a sequence for student to register courses

1.2.2

Let $n < 8\lg n$, we can see that for $n \leq 64$, this holds.

1.2.3

n	$100n^2$	2^n
10	10000	1024
13	16900	8192
14	19600	16384
15	22500	32768

Starts from 15, $100n^2$ is faster than 2^n;

1.3 Problem

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$	2^{10^6}	$2^{10^7 \cdot 6}$	$2^{10^8 \cdot 36}$	$2^{10^8 \cdot 864}$	$2^{10^8 \cdot 25920}$	$2^{10^8 \cdot 311040}$	$2^{10^8 \cdot 31104000}$
$\sqrt{n}$	10^{12}	$36 \cdot 10^{14}$	$1296 \cdot 10^{16}$	$746496 \cdot 10^{16}$	$6718464 \cdot 10^{18}$	$967458816 * 10^{18}$	$967458816 \cdot 10^{22}$
n	10^6	$10^7 \cdot 6$	$10^8 \cdot 36$	$10^8 \cdot 864$	$10^8 \cdot 25920$	$10^8 \cdot 311040$	$10^{10} \cdot 311040$
$n\lg n$	62746	2801417	133378058	2755147513	71870856404	797633893349	68654697441062
n^2	1000	7745	60000	293938	1609968	5615692	56175382
n^3	100	391	1532	4420	13736	31593	146677
$2n$	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Chapter 2

Getting Started

2.1 Exercises

2.1.1

1. **31**, 41, 59, 26, 41, 58
2. **31, 41**, 59, 26, 41, 58
3. **31, 41, 59**, 26, 41, 58
4. **31, 41, 26, 59**, 41, 58 -> **31, 26, 41, 59**, 41, 58 -> **26, 31, 41, 59**, 41, 58
5. **26, 31, 41, 41, 59**, 58
6. **26 ,31, 41, 41, 58, 59**

2.1.2

INSERTION-SORT-DECREASE(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      i = j − 1
4      while i > 0 and A[i] < key
5          A[i] = A[i + 1]
6          i = i − 1
7      A[i + 1] = key
```

2.1.3

LINEAR-SEARCH(A, υ)

```
1  for i = 1 to A.length
2      if A[i] == υ
3          return i
4  return NIL
```

Given the **loop invariant**:
At the start of each iteration of the for **loop of lines 1-3, the subarray** $A[1..i-1]$ **consists of the elements that are all not equal to** υ
The three properties:

Initialization We start by $i = 1$, the subarrary $A[1..i-1]$ contains no element. So this holds

Maintenance Before every cycle, we know that elements in the subarray $A[1..i-1]$ are all different from υ, if $A[i] = \upsilon$, then we return i. Otherwise, we go to the next cycle. Because $A[i]$ is different from υ, it is maintained.

Termination The loop terminates:
1.when $i > A.length$, which is $i = A.lengh + 1$, thus the subarrary $A[1..A.length]$ contains elements different from υ, and we return NIL.
2.We found an index i, s.t. $A[i] = \upsilon$ and return i .

2.1.4

Input: Two bit 1-0 array $A[a_1, a_2, ..., a_n]$ and $B[b_1, b_2, ..., b_n]$ which represent two integer value $A^{'}, B^{'}$ in binary.

Output: A bit 1-0 array $C[c_1, c_2, ..., c_{n+1}]$, representing a integer value $C^{'}$, s.t. $C^{'} = A^{'} + B^{'}$.

ADD-INTEGER(A, B)
1 $C = new\ integer[A.length]$
2 $carry = 0$
3 **for** $i = 1$ **to** $A.length$
4 $\quad C[i] = (A[i] + B[i] + carry) \mod 2$
5 $\quad carry = (A[i] + B[i] + carry)/2$
6 $C[A.length + 1] = carry$
7 **return** C

2.2 Exercises

2.2.1

$\Theta(n^3)$.

2.2.2

SELECTION-SORT(A)
1 **for** $i = 1$ **to** $A.length - 1$
2 $\quad c = A[i]$
3 $\quad m = i$
4 $\quad$ **for** $j = i + 1$ **to** $A.length$
5 $\quad\quad$ **if** $A[j] < A[m]$
6 $\quad\quad\quad m = j$
7 $\quad A[i] = A[m]$
8 $\quad A[m] = c$

Given the **loop invariant**:
At the start of each iteration of the for **loop of lines 1-3, the subarray** $A[1..i-1]$ **consists of the elements that are in the original array** A**, but in sorted order, and they are all smaller than the other elements in the original array**.
The three properties:

Initialization We start by $i = 1$, the subarrary $A[1..i-1]$ contains no element. So this holds

Maintenance Before every cycle, we know that elements in the subarray $A[1..i-1]$ are all smaller than elements in the subarray $A[i..A.length]$. Next, we choose the smallest element from that subarray, swap it with $A[i]$, note that this element is bigger than the elements in subarray $A[1..i-1]$. Thus, the property is maintained.

Termination The loop terminates when $i = A.length$, thus the subarrary $A[1..A.length-1]$ contains elements that are in the original array but in sorted order, and they are smaller than $A[i]$, Observing that, the whole array A is in sorted order.Thus, this algorithm is correct.

From the loop invariant proof of this algorithm, we know why it stop at $n-1$. And from simply analysis, it's not difficult to find out that both the worst-case and best-case are bounded to $\Theta(n^2)$.

2.2.3

suppose we have n elements in total, then the average number of elements that we need to check is,

$$t = \frac{1+2+\cdots+n}{n} = \frac{n+1}{2}$$

No doubt , in the worst case

$$t = n$$

In both cases, $t = \Theta(n)$

2.2.4

Design a specific check method to check if all is done.

2.3 Exercises

2.3.1

3 9 26 38 41 49 52 57

3 26 41 52 9 38 49 57

3 41 26 52 38 57 9 49

3 41 52 26 38 57 9 49

2.3.2

NEW-MERGE-SORT(A,p,q,r)

```
1  n1 = q − p + 1
2  n2 = r − q
3  let L[1..n1 + 1]andR[1..n2 + 1] be new arrays
4  for i = 1 to n1
5      L[i] = A[p + i − 1]
6  for i = 1 to n2
7      R[i] = A[q + i]
8  i = 1
9  j = 1
10 while i ≤ n1 and j ≤ n2
11     if L[i] ≤ R[j]
12         A[p] = L[i]
13         i = i + 1
14     else
15         A[p] = R[j]
16         j = j + 1
17     p = p + 1
18 while i ≤ n1
19     A[p] = L[i]
20     i = i + 1
21     p = p + 1
22 while j ≤ n2
23     A[p] = R[j]
24     j = j + 1
25     p = p + 1
```

2.3.3

Basis when $k = 1, n = 2$, $T(n) = 2 = 2 \times \lg 2$ holds.

Induction Suppose, for $n = 2^k, k \geq 1$, $T(n) = n \lg n$ holds, then we can derive that, for $k' = k+1, n' = 2^{k'} = 2^{k+1}$

$$\begin{aligned} T(n') &= 2T(n'/2) + n' \\ &= 2n \lg n + n' \\ &= k2^{k+1} + 2^{k+1} \\ &= (k+1) \times 2^{k+1} \\ &= n' \lg n' \end{aligned}$$

2.3.4

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n-1) + C(n-1) & \text{otherwise} \end{cases}$$

2.3.5

BINARY-SEARCH($A, low, high, key$)

```
1  if low > high
2      return NIL
3  mid = (low + high)/2
4  if A[mid] == key
5      return mid
6  elseif A[mid] > key
7      return BINARY-SEARCH(A, low, mid − 1, key)
8  else
9      return BINARY-SEARCH(A, mid + 1, high, key)
```

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

Proof. Each time, we check if the mid-element equal to key, this cost constant time. If not, we halve the problem size unless there is only one element to check. For the worst case, we halve the problem every time and try to find the element that equals to the key. We can only do this for at most $\lg n$ times and each time's cost is constant, therefore, the worst-case running time of binary search is $\Theta(\lg n)$. □

2.3.6

Nope.

If the cost for comparison between elements is minimal compared with that of move the elements, then binary-search will not help much because we still have to move the elements by $\Theta(n)$ cost in the worst-case.
If the cost for comparison is dominant in overall cost then binary-search can make great improvements on INSERTION-SORT algorithm. But the time complexity is still $\Theta(n \lg n)$ due to the swapping of elements.

2.3.7

We can sort the set S in $\Theta(n \lg n)$ time and search in $\Theta(n)$ time, that gives us an overall $\Theta(n \lg n)$ algorithm. The linear search algorithm is given below (S is sorted).

SEARCH-SUM(S, sum)

```
1  left = 1
2  right = S.length
3  while left < right
4      s = S[left] + S[right]
5      if s == sum
6          return TRUE
7      elseif s > sum
8          right = right − 1
9      else
10         left = left + 1
11 return FALSE
```

2.4 Problem

2.4.1 Insertion sort on small arrays in merge sort

a sort a list of length k : $\Theta(k^2)$

sort n/k list of length k : $\Theta(n/k * k^2) = \Theta(nk)$

Chapter 3

Growth of Functions

3.1 Exercises

3.1.1

Because $f(n) and g(n)$ are asymptotically nonnegative functions, thus:

$$\exists n_1 s.t. f(n) > 0 (n \geq n_1)$$
$$\exists n_2 s.t. g(n) > 0 (n \geq n_2)$$

Let $n_0 = max\{n_1, n_2\}$,
when $n > n_0$:

$$f(n) \leq max\{f(n), g(n)\}$$
$$g(n) \leq max\{f(n), g(n)\}$$

thus:

$$\frac{f(n)+g(n)}{2} \leq max\{(f(n), g(n)\} \tag{3.1.1}$$

holds. And

$$max\{f(n), g(n)\} \leq f(n)+g(n) \tag{3.1.2}$$

(3.1.1) and (3.1.2) states that:

$$\exists n_0 : \frac{f(n)+g(n)}{2} \leq max\{f(n), g(n)\} \leq f(n)+g(n) (n \geq n_0)$$

therefore:

$$max\{f(n), g(n)\} = \Theta(f(n)+g(n))$$

3.1.2

To show $(n+a)^b = \Theta(n^b)$, we need to find positive constants c_1, c_2, n_0 such that, when $n \geq n_0$, $0 \leq c_1 n^b \leq (n+a)^b \leq c_2 n^b$ holds.
We divide both sides of the inequality by n^b
since $n > 0$ and $b > 0$, we have:

$$c_1 \leq (1+\frac{a}{n})^b \leq c_2$$

Let $n_0 = 2[|a|]$, then when $n > n_0$

$$(\frac{1}{2})^b < (1+\frac{a}{n})^b < (\frac{3}{2})^b$$

holds.
Thus we can pick $c_1 = (\frac{1}{4})^b$,$c_2 = 2^b$

$$\begin{aligned} &\text{When } n > n_0, \\ 0 \le c_1 n^b \le (n+a)^b &\le c_2 n^b \\ &thus, \\ (n+a)^b &= \Theta(n^b) \end{aligned}$$

3.1.3

Let $T(n)$ denote the running time of algorithm A. When we say $T(n)$ is at least $O(n^2)$, we mean,
$\exists f(n)$,and two positive constant $c, n_0, s.t. 0 \le f(n) \le cn^2 (n > n_0)$, and $T(n) \ge f(n) (n > n_0)$. What is meaningless is that, $f(n)$ is not a determined function. Consequently, we know nothing about whether $T(n) = \Omega(n^2)$, $T(n) = \Theta(n^2)$ or $T(n) = O(n^2)$. We have no idea what T(n) is going to be.

3.1.4

$\forall n > 0, 0 < 2^{n+1} \le 2 \cdot 2^n$holds, thus $2^{n+1} = O(2^n)$
But $2^{2n} \ne O(2^n)$ since 2^{2n} always exceeds 2^n at some point of n_0.

3.1.5

Proof. $\implies$ Because $f(n) = \Theta(g(n))$, $\exists n_0, c_1, c_2 > 0$ s.t. when $n \ge n_0$

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$

this suffices that $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$

$\impliedby$ When $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$
$\exists n_1, c_1, s.t.$, when $n \ge n_1$, $f(n) \ge c_1 g(n) \ge 0$
$\exists n_2, c_2, s.t.$, when $n \ge n_2$, $0 \le f(n) \le c_2 g(n)$
Let $n_0 = max\{n_1, n_2\}$, thus when $n \ge n_0$

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$

So, $f(n) = \Theta(g(n))$

□

3.1.6

Proof. Let $T(n)$ be the running time of an algorithm, $T_\omega(n)$ be the worst-case running time, $T_b(n)$ be the best-case running time. Note that

$$T_b(n) \le T(n) \le T_\omega(n)$$

If $T_b(n) = \Omega(g(n))$ and $T_\omega(n) = O(g(n))$, no doubt $T(n) = \Theta(g(n))$
On the other hand, if $T(n) = \Theta(g(n))$, then $T(n) = \Omega(g(n))$ and $T(n) = O(g(n))$ must holds by $Theorem 3.1$. And $T_b(n)$ and $T_\omega(n)$ are just corner cases of $T(n)$, thus $T_b(n) = \Omega(g(n))$ and $T_\omega(n) = O(g(n))$ are true. □

3.1.7

Assume it's not the empty set. As a result, there exists such $f(n)$ that ,for any positive constant $c > 0$, there exists a constant n_0 such that $0 \leq f(n) \leq cg(n)$ and $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$, this cannot be true because $f(n)$ must satisfy for all positive c.
Therefore, $o(g(n)) \cap \omega(g(n))$ is the empty set.

3.1.8

$$\begin{aligned}\Omega(g(n,m)) = \{f(n,m) : &\text{ there exist positive constants } c, n_0, \text{and } m_0 \\ &\text{such that } 0 \leq cg(n,m) \leq f(n,m) \\ &\text{for all } n \geq n_0 \text{ or } m \geq m_0\}\end{aligned}$$

$$\begin{aligned}\Theta(g(n,m)) = \{f(n,m) : &\text{ there exist positive constants } c_1, c_2, n_0, \text{and } m_0 \\ &\text{such that } 0 \leq c_1 g(n,m) \leq f(n,m) \leq c_2 g(n,m) \\ &\text{for all } n \geq n_0 \text{ or } m \geq m_0\}\end{aligned}$$

3.2 Exercises

3.2.1

Let $0 \leq m \leq n$, since $f(n)$ and $g(n)$ are monotonically increasing functions,

$$\begin{aligned} f(m) &\leq f(n) \\ g(m) &\leq g(n) \end{aligned}$$

therefore,

$$\begin{aligned} f(m) + g(m) &\leq f(n) + g(n) \\ f(g(m)) &\leq f(g(n)) \end{aligned}$$

For the last one,

$$\begin{aligned} f(n)g(n) - f(m)g(m) &= f(n)g(n) - f(m)g(n) + f(m)g(n) - f(m)g(m) \\ &= [f(n) - f(m)]g(n) + [g(n) - g(m)]f(m) \\ &\geq 0 \end{aligned}$$

Done!

3.2.2

The Equation is,

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

take logarithm on both sides,

$$log_b c \; lna = log_b a \; lnc$$

then, use equation(3.15),

$$\frac{lnc}{lnb} lna = \frac{lna}{lnb} lnc$$

we get,

$$lnc \; lna = lna \; lnc$$

Q.E.D.

3.2.3

use *Stirling's approximation*,

$$\begin{aligned} lg(n!) &= lg\sqrt{2\pi n} + nlg\frac{n}{e} + lg(1+\Theta(\frac{1}{n})) \\ &= \Theta(lgn) + \Theta(nlgn) + \Theta(\frac{1}{n}) \\ &= \Theta(nlgn) \end{aligned}$$

for $n! = \omega(2^n)$:

$$\forall n > 3: \;\; 0 < 2^n = 2 \cdot 2 \cdots 2 < n(n-1)(n-2)\cdots 1 = n! \implies n! = \omega(2^n)$$

for $n! = o(n^n)$:

$$\forall n > 1: \;\; 0 < n! = n(n-1)\cdots 1 < n \cdot n \cdots n = n^n \implies n! = o(n^n)$$

3.2.4

if a function $f(n)$ is polynomially bounded, then there exist positive constants c, k, n_0, $f(n) = O(n^k)$such that when $n \geq n_0$, $0 \leq f(n) \leq cn^k$, take logarithm on the right part of the inequality,

$$\begin{aligned} \lg f(n) &\leq k\lg n + \lg c \\ \lg f(n) &= O(\lg n) \end{aligned}$$

for the ceiling mark we know,

$$\lg n \leq \lceil \lg n \rceil < \lg n + 1 < 2\lg n$$

so $\lceil \lg n \rceil = \Theta(\lg n)$ Therefore, using Eq.(3.19),

$$\begin{aligned} \lg\lceil \lg n \rceil ! &= \Theta(\lceil \lg n \rceil \lg(\lceil \lg n \rceil)) \\ &= \Theta(\lg n \lg\lg n) \\ &= \omega(\lg n) \end{aligned}$$

and,

$$\begin{aligned} \lg\lceil \lg\lg n \rceil ! &= \Theta(\lceil \lg\lg n \rceil \lg(\lceil \lg\lg n \rceil)) \\ &= \Theta(\lg\lg n \lg\lg\lg n) \\ &= \Theta((\lg\lg n)^2) \\ &= \Theta(\lg^2 \lg n) \\ &= o(\lg n) \\ &= O(\lg n) \end{aligned}$$

therefore, the function $\lceil \lg n \rceil !$ is not polynomially bounded, but $\lceil \lg\lg n \rceil !$ is.

3.2.5

The latter one. Let $2^m = n$

$$\begin{aligned} \lg\lg^* n = \lg\lg^* 2^m &= \lg(1 + \lg^* m) \\ \lg^* \lg n = \lg^* \lg 2^m &= \lg^* m = \omega(1 + \lg^* m) \end{aligned}$$

3.2.6

the solution of this equation are

$$x_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

which are exactly ϕ and $\hat{\phi}$.

3.2.7

Base Case when $n = 1$, $Fibonacci(1) = 1$, while $F_1 = 1$ satisfies.

Induction Steps for $1, 2, 3, ...k-1$ the equality holds, then for $n = k$, by definition

$$\begin{aligned} F_k = F_{k-1} + F_{k-2} &= \frac{\phi^{k-1} - \hat{\phi}^{k-1}}{\sqrt{5}} + \frac{\phi^{k-2} - \hat{\phi}^{k-2}}{\sqrt{5}} \\ &= \frac{1}{\sqrt{5}}[(\frac{3+\sqrt{5}}{2})\phi^{k-2} - (\frac{3-\sqrt{5}}{2})\hat{\phi}^{k-2}] \\ &= \frac{1}{\sqrt{5}}[\phi^2\phi^{k-2} - \hat{\phi}^2\hat{\phi}^{k-2}] \\ &= \frac{1}{\sqrt{5}}[\phi^k - \hat{\phi}^k] \end{aligned}$$

DONE!

3.2.8

Because $k \ln k = \Theta(n)$,

$$n = \Theta(k \ln k)$$

therefore,

$$\begin{aligned} \frac{n}{\ln n} &= \Theta(\frac{k \ln k}{\ln n}) \\ &= \Theta(\frac{k \ln k}{\ln(k \ln k)}) \\ &= \Theta(\frac{k \ln k}{\ln k + \ln \ln k}) \\ &= \Theta(k) \end{aligned}$$

thus,

$$k = \Theta(\frac{n}{\ln n})$$

3.3 Problems

3.3.1

a. To show that $p(n) = O(n^k)$, we have to find the constans n_0, c such that when $n \geq n_0$, $p(n) \leq cn^k$.

Let $c = a_d + q$, to find the n_0, let

$$p(n) = a_0 + a_1 n + \cdots + a_d n^d \leq cn^k$$

divide both sides by n^k,

$$\frac{1}{n^k}(a_0 + a_1 n + \cdots + a_{d-1} n^{d-1}) + a_d n^{d-k} \leq a_d + q$$

since $k \geq d$, we know $a_d n^{d-k} \leq a_d$, now we only have to show that

$$\frac{1}{n^k}(a_0 + a_1 n + \cdots + a_{d-1} n^{d-1}) \leq q$$

Let $a_m = max(|a_0|, |a_1|, ..., |a_{d-1}|)$, and $q = 1$, we only need to show

$$\frac{a_m}{n^k}(1 + n + ... + n^{d-1}) \leq 1$$

$$\frac{a_m}{n^k}\frac{n^d - 1}{n-1} \leq 1$$

because

$$\frac{n^d - 1}{n-1} \leq \frac{n^d}{n/2} \quad (n \geq 2)$$

Let

$$\frac{a_m}{n^k}\frac{n^d - 1}{n-1} \leq \frac{a_m}{n^k}\frac{n^d}{n/2}$$
$$\leq 1$$

we know

$$n \geq \sqrt[k+1-d]{a_m}$$

choose $n_0 = max(\sqrt[k+1-d]{a_m}, 2)$ and $c = a_d + 1$, we are done.

the other 4 are proved similar processes.

3.3.2

a $\lg^k n = o(n^\varepsilon) = O(n^\varepsilon)$

b $n^k = o(c^n) = O(c^n)$

c No certain relations.

d $2^n = \omega(2^{n/2}) = \Omega(2^{n/2})$

e $n^{\lg c} = O(c^{\lg n}), n^{\lg c} = \Omega(c^{\lg n}), n^{\lg c} = \Theta(c^{\lg n})$. In fact, $c^{\lg n} = n^{\lg c}$

3.3.3

Rank↓

1 $2^{2^{n+1}}$

2 2^{2^n}

3 $(n+1)!$

4 $n!$

5 e^n

6 $n \cdot 2^n$

7 2^n

8 $(3/2)^n$

9 $(\lg n)!$

10 $n^{\lg\lg n}$, $(\lg n)^{\lg n}$

11 n^3

12 $4^{\lg n}$, n^2

13 $\lg(n!)$, $n\lg n$

14 $2^{\lg n}$, n

15 $(\sqrt{2})^{\lg n}$

16 $2^{\sqrt{2\lg n}}$

17 $\lg^2 n$

18 $\ln n$

19 $\sqrt{\lg n}$

20 $\ln\ln n$

21 $2^{\lg^* n}$

22 $\lg^* n$, $\lg^*(\lg n)$

23 $\lg(\lg^* n)$

24 $n^{1/\lg n}$, 1

Example: $f(n) = 2^{2^{n+1}} sin(n)$

3.3.4

a wrong. Let$f(n) = n, g(n) = n^2$

b wrong. Choose the same $f(n), g(n)$ as a

c correct.

Proof. $f(n) = O(g(n))$ means there exists constants n_0, c such that, when $n \geq n_0$, $0 \leq f(n) \leq cg(n)$, because $f(n) \geq 1$ and $\lg(g(n)) \geq 1$ for all sufficiently large n,

$$0 \leq \lg f(n) \leq \lg(g(n)) + \lg c$$

thus, $\lg f(n) = O(\lg(g(n)))$ □

d incorrect. $f(n) = 2n, g(n) = n$

e wrong. Let $f(n) = 1/n$.

f correct.

g wrong. Let $f(n) = 2^n$

h correct. Let $g(n) = o(f(n))$, thus, for any positive constant c, $0 \leq g(n) < cf(n)$ when n exceed some n_0, let $c = 1$, then

$$f(n) \leq f(n) + g(n) < 2f(n)$$

therefore,

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

3.3.5

a for

$$f(n) \geq cg(n)$$

if there are finite many integers n satisfy this inequality, then choose n_0 to be the max among them.$f(n) = O(g(n))$

if there are infinite many, then $f(n) = \overset{\infty}{\Omega}(g(n))$

Both holds if $f(n) = g(n)$.

b We can use this symbol to describe some unstable running time algorithms like $n^3 sin(n)$.

but things may get unclear to some points.

c Becomes "implies".

d

$$\widetilde{\Omega}(g(n)) = \{f(n) : \text{there exist positive constants } c, k \text{ and } n_0 \text{ such that } f(n) \geq cg(n)\lg^k(n) \geq 0 \text{ for all } n \geq n_0\}$$

$$\widetilde{\Theta}(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, k_1, k_2 \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n)\lg^{k_1} n \leq f(n) \leq c_2 g(n)\lg^{k_2}(n) \text{ for all } n \geq n_0\}$$

Proof. $\Longrightarrow$ Because $f(n) = \widetilde{\Theta}(g(n))$, $\exists n_0, c_1, c_2, k_1, k_2 > 0$ s.t. when $n \geq n_0$

$$0 \leq c_1 g(n) \lg^{k_1} n \leq f(n) \leq c_2 g(n) \lg^{k_2} n$$

this suffices that $f(n) = \widetilde{\Omega}(g(n))$ and $f(n) = \widetilde{O}(g(n))$

$\Longleftarrow$ When $f(n) = \widetilde{\Omega}(g(n))$ and $f(n) = \widetilde{O}(g(n))$
$\exists n_1, c_1, k_1 s.t.$, when $n \geq n_1$, $f(n) \geq c_1 g(n) \lg^{k_1} n \geq 0$
$\exists n_2, c_2, k_2 s.t.$, when $n \geq n_2$, $0 \leq f(n) \leq c_2 g(n) \lg^{k_2} n$
Let $n_0 = max\{n_1, n_2\}$, thus when $n \geq n_0$

$$0 \leq c_1 g(n) \lg^{k_1} n \leq f(n) \leq c_2 g(n) \lg^{k_2} n$$

So, $f(n) = \widetilde{\Theta}(g(n))$

□

3.3.6

	$f(n)$	c	$f_c^*(n)$
a.	$n-1$	0	$\Theta(n)$
b.	$\lg n$	1	$\Theta(\lg^* n)$
c.	$n/2$	1	$\Theta(\lg n)$
d.	$n/2$	2	$\Theta(\lg n)$
e.	$\sqrt{n}$	2	$\Theta(\lg\lg n)$
f.	$\sqrt{n}$	1	*None*
g.	$n^{\frac{1}{3}}$	2	$\log_3 \lg n$
h.	$n/\lg n$	2	$\omega(\log_{\frac{1}{1-\varepsilon}} \lg n), o(\lg n)$

Notes :

f for $f(n) = \sqrt{n}$, we know $\lim_{n\to\infty} \sqrt[n]{n} = 1$, but for finite n, k, $n^{\frac{1}{2^k}} > 1$

h ε is a positive constant.