## Synopsis of Algorithm & Data Structure Analysis

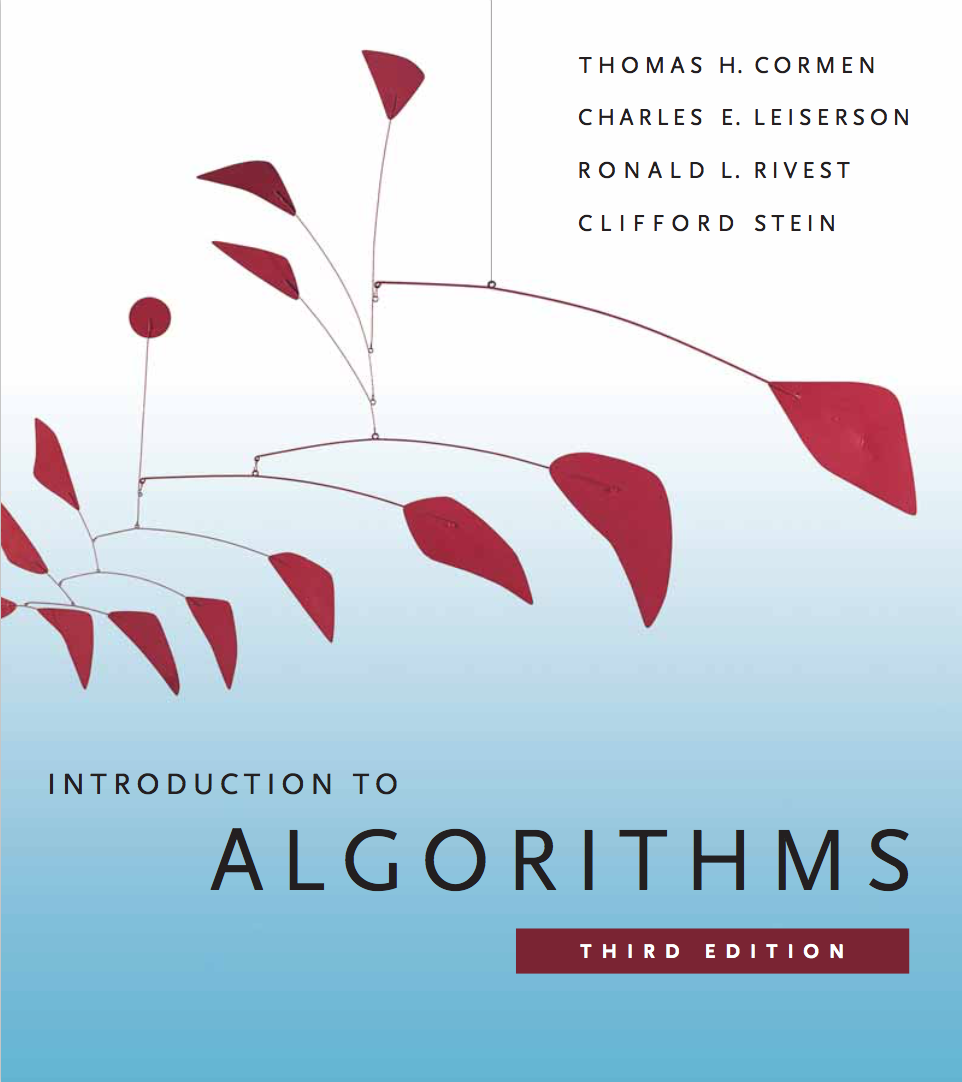
## (Semester 2 - 2017)

The whole course requires at least 12 sessions to be complete. (Each session is on a two-hour basis).

By the time students finish this class, they'll know how to apply most of algorithms to such questions as Karatsuba, priority queues and heaps, binary search, AVL tree rotations, hashing, graphs representation, DFS and BFS, and many other questions. They'll also know how to select the right and appropriate words and phrases to answer questions needed to explain in details.

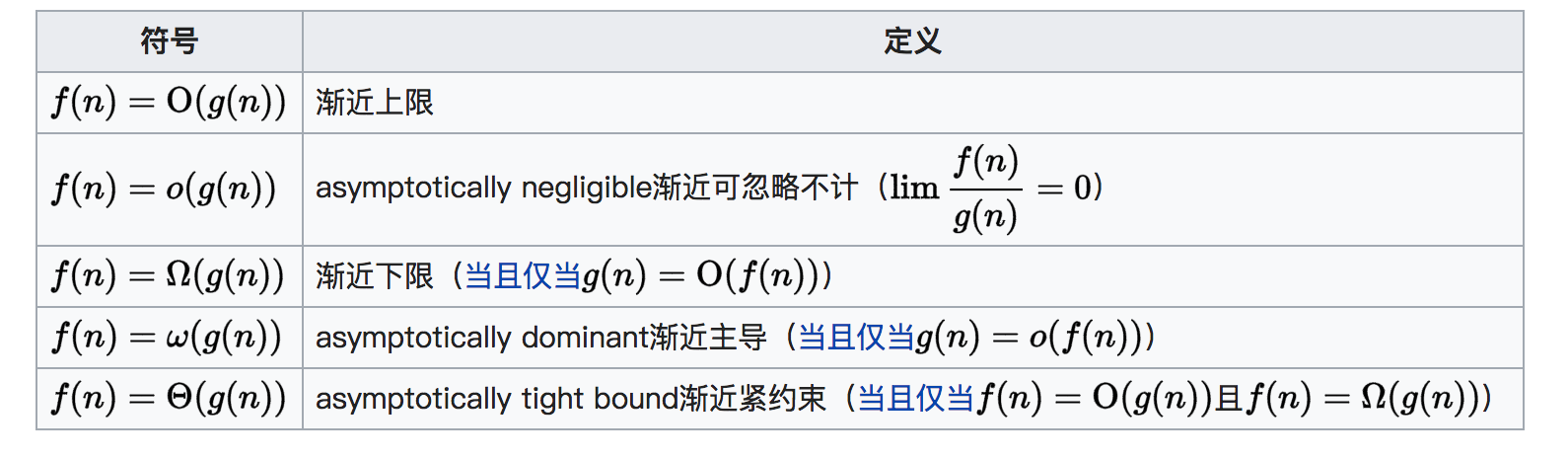
**Corresponding textbook:**

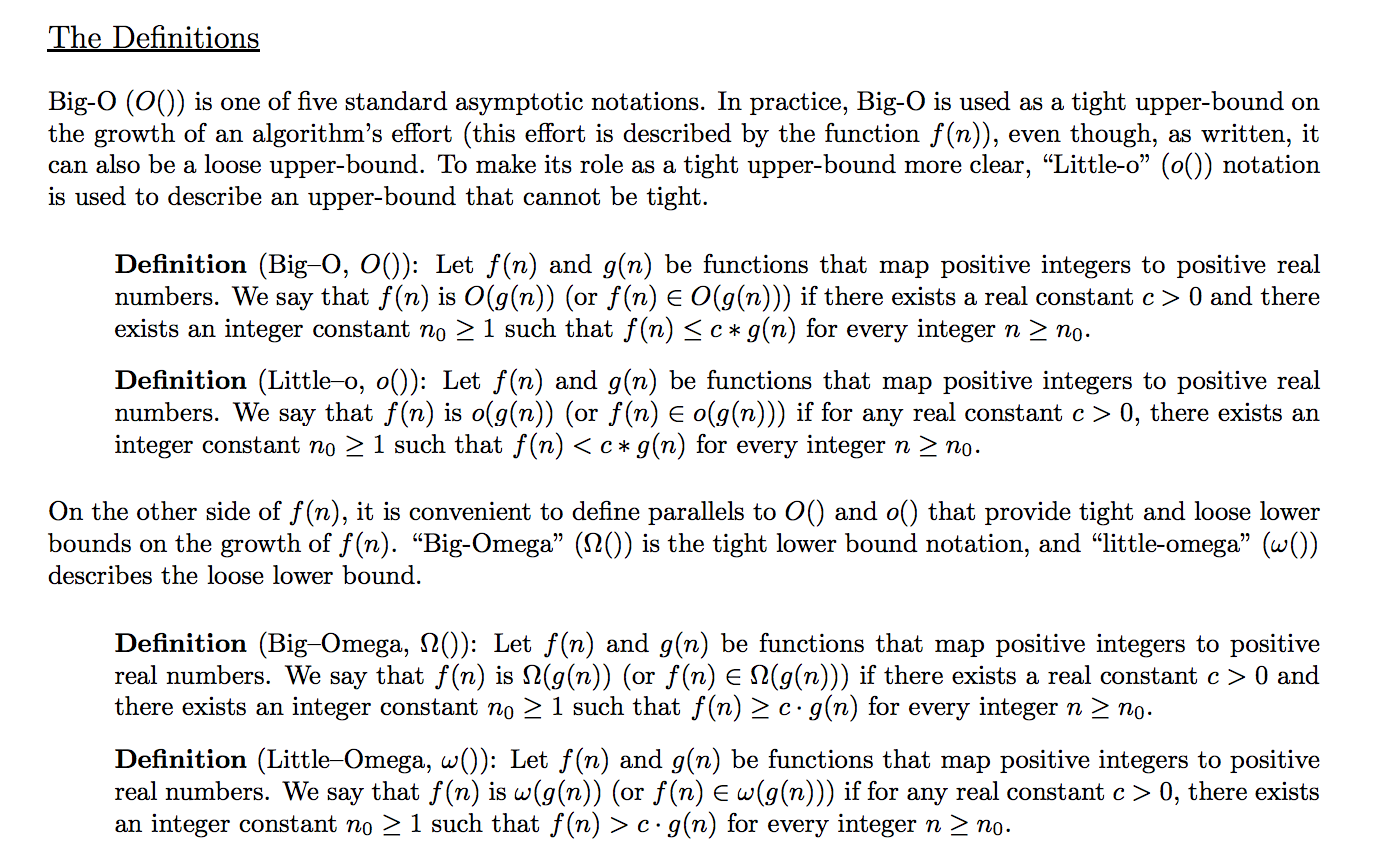
The textbook for this course is

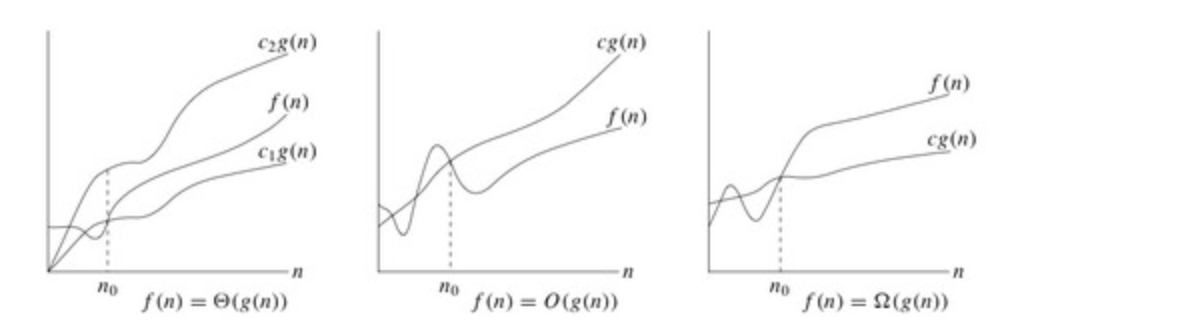
****

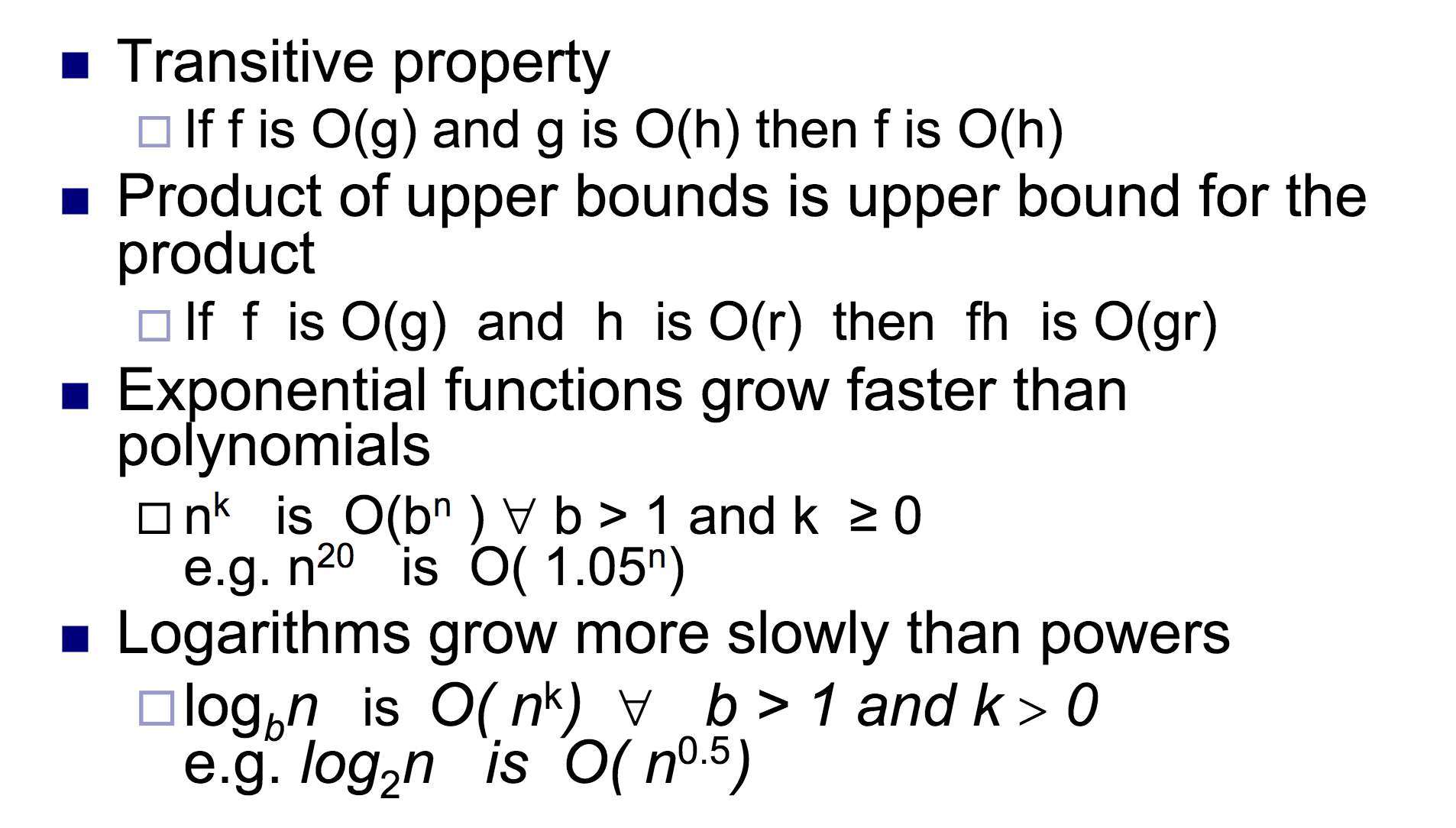
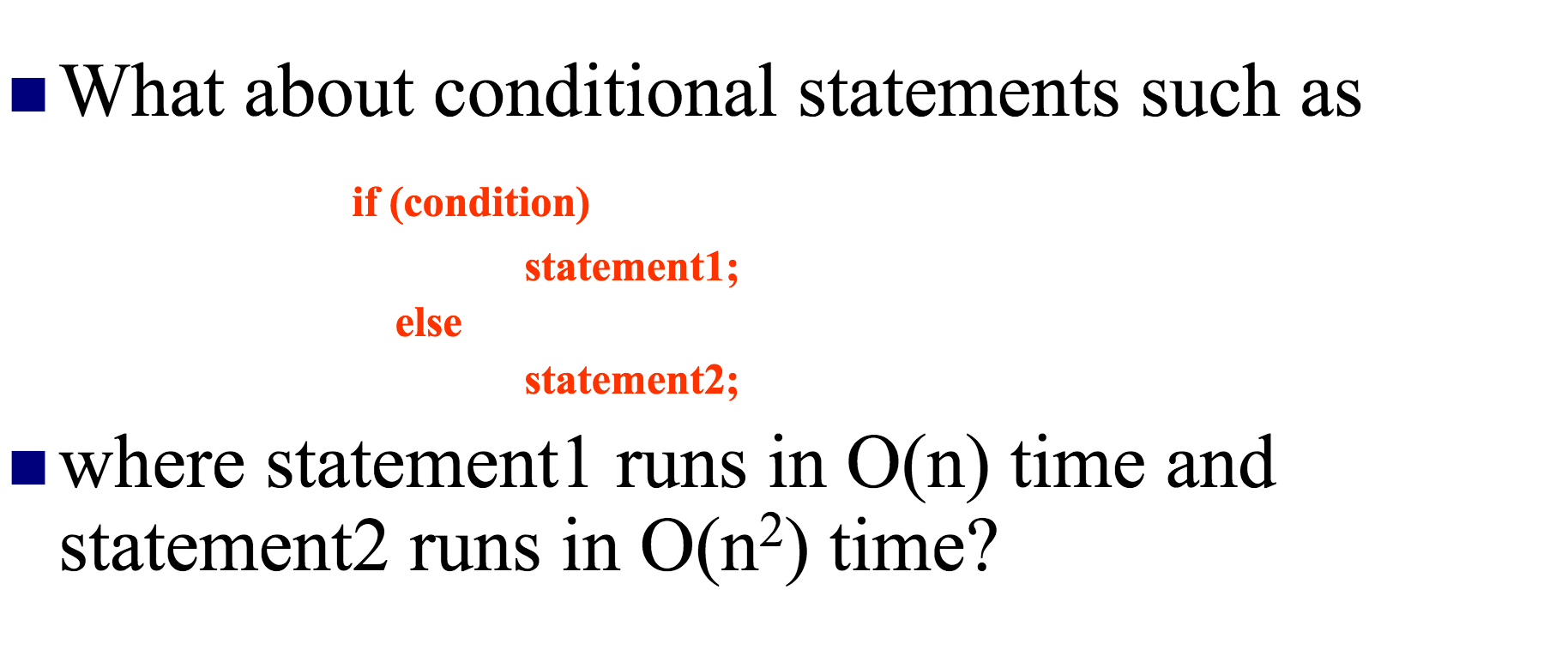
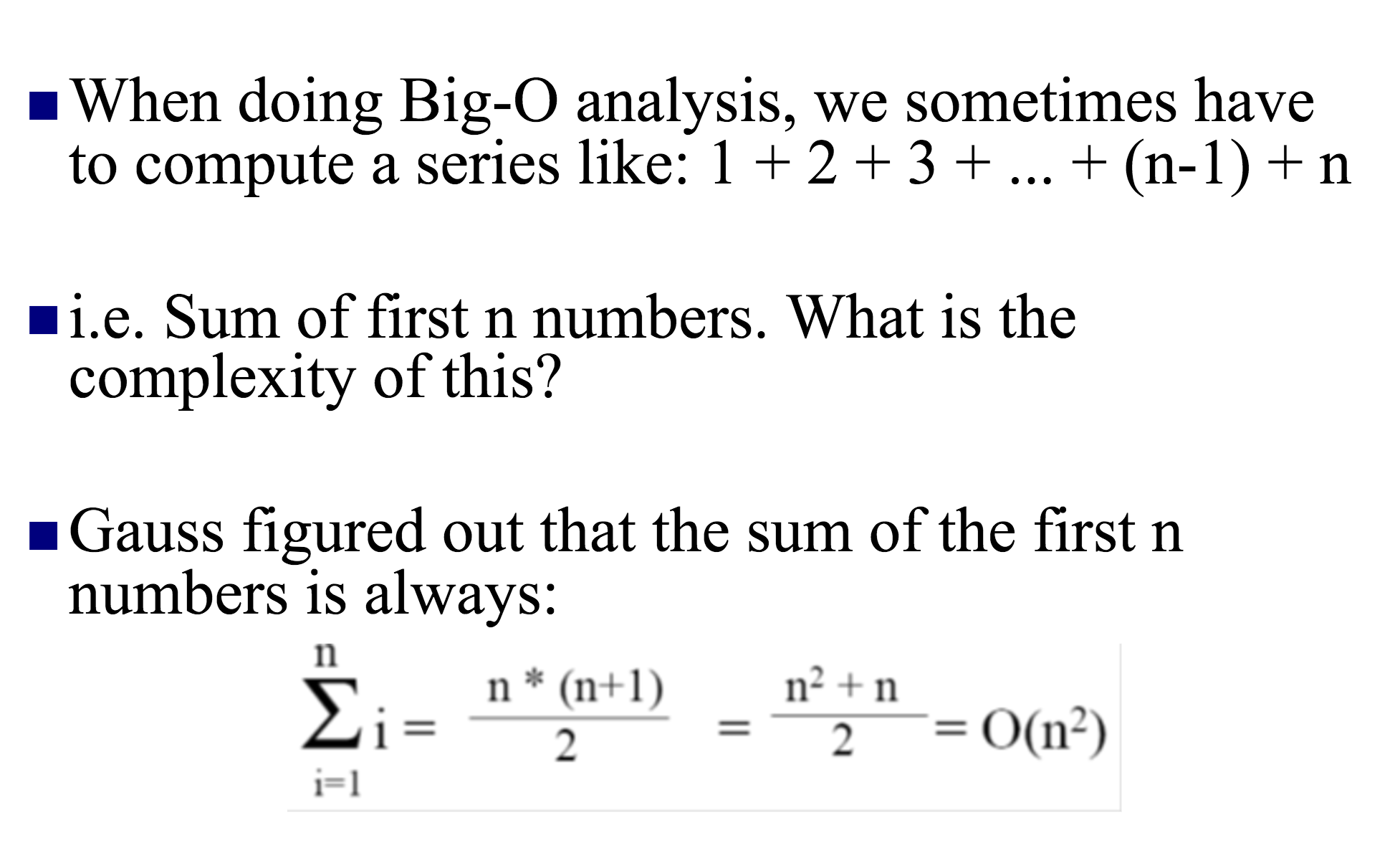
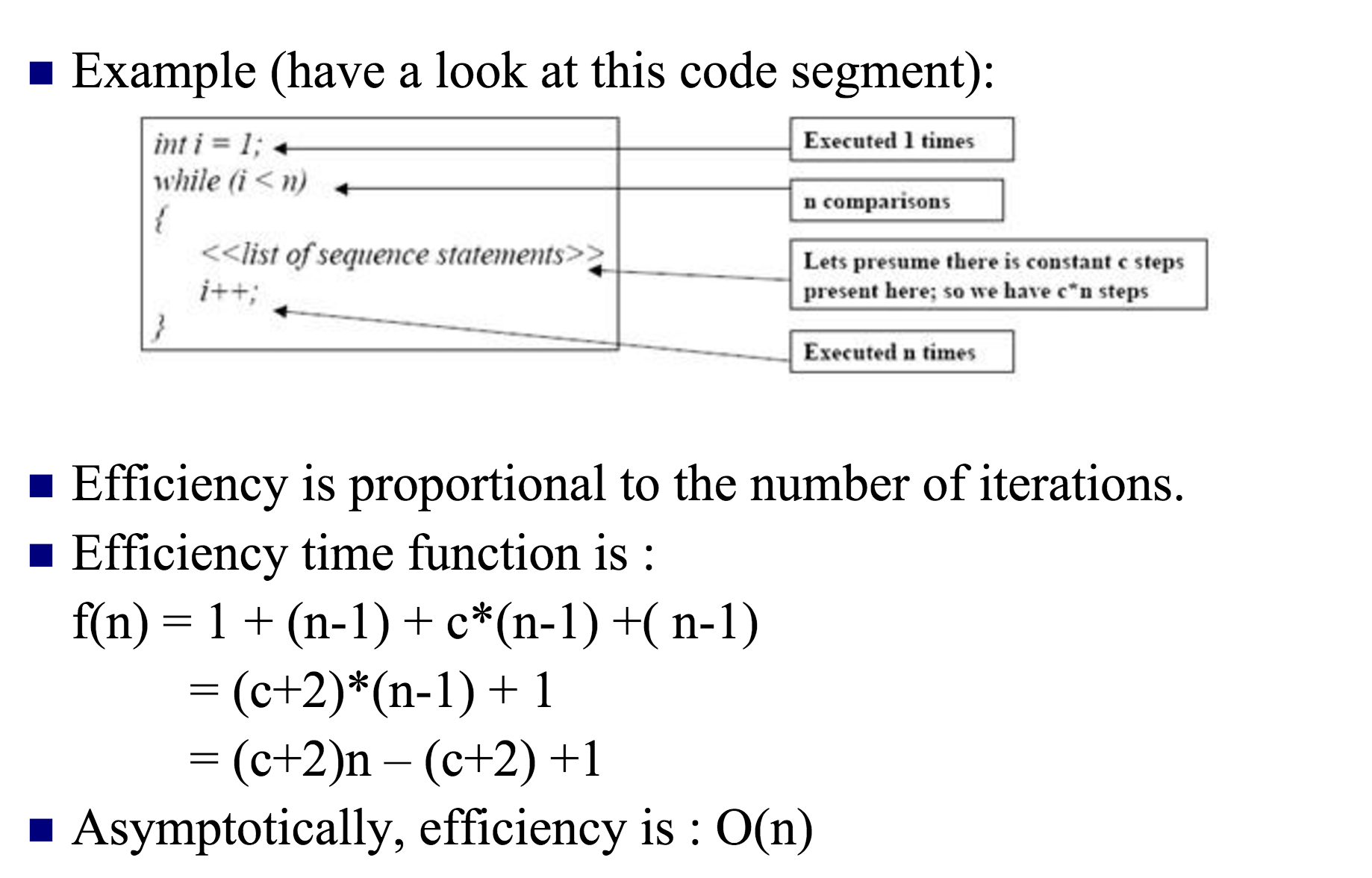
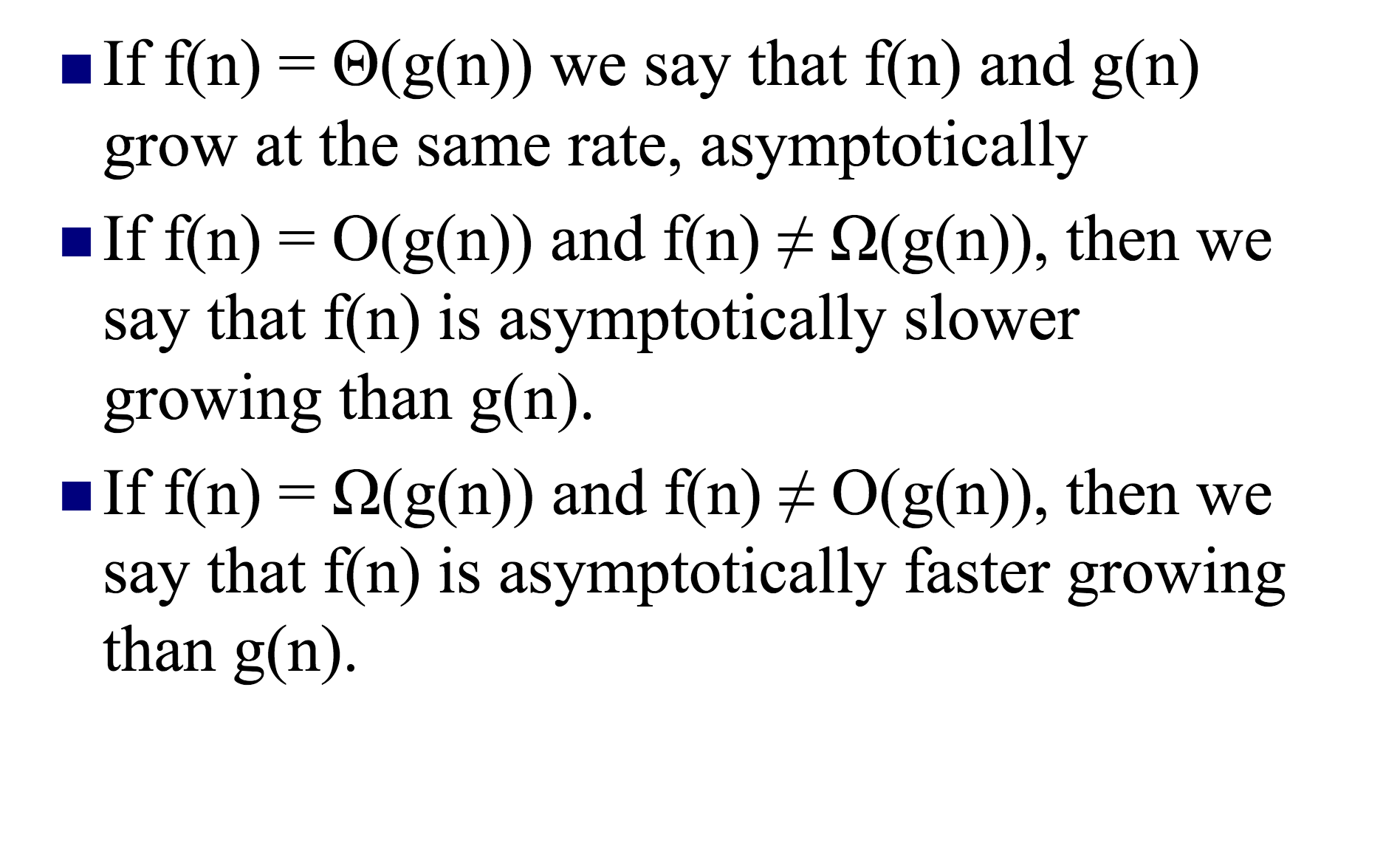
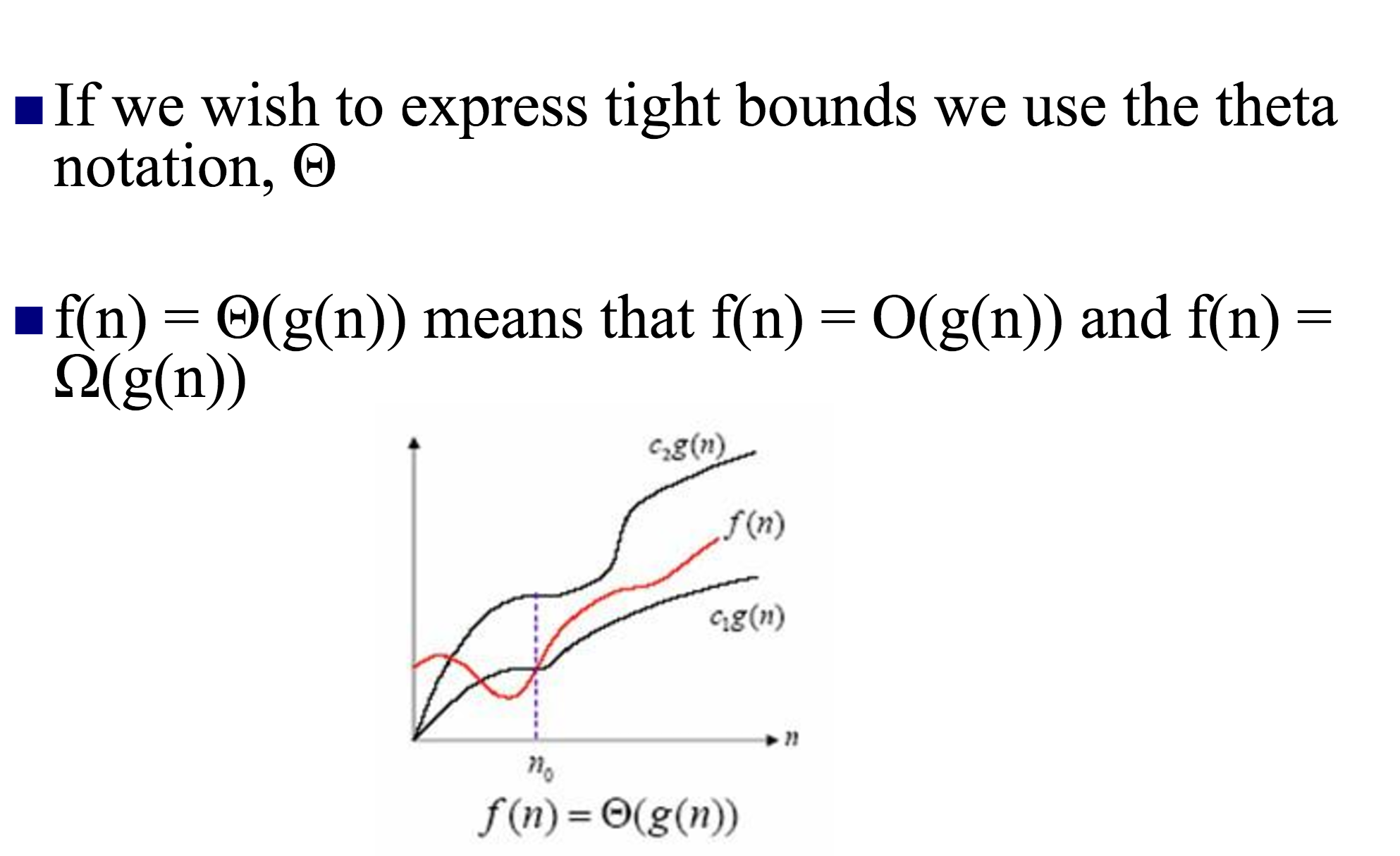
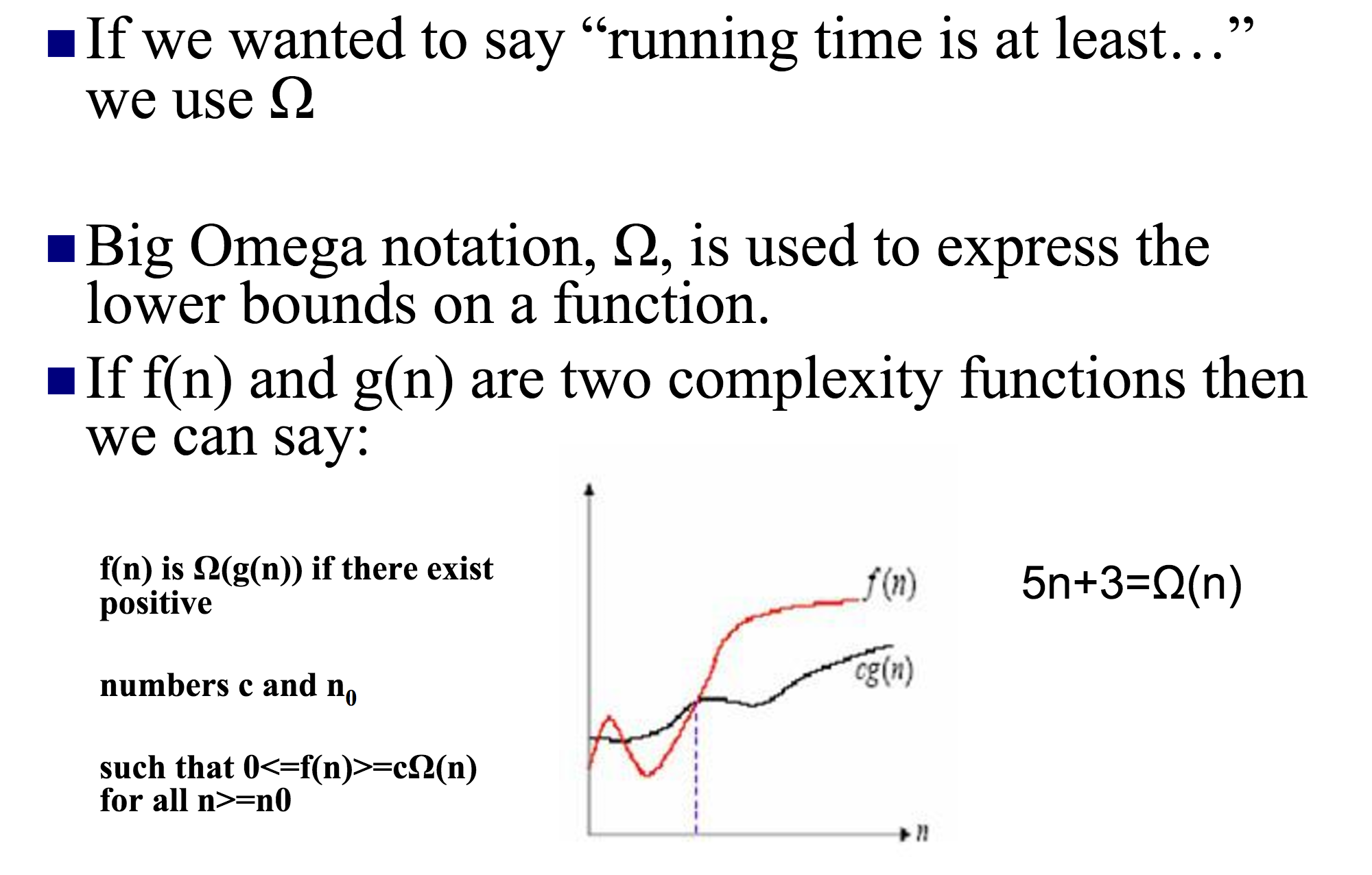
Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, Third Edition, MIT Press.

**big O**

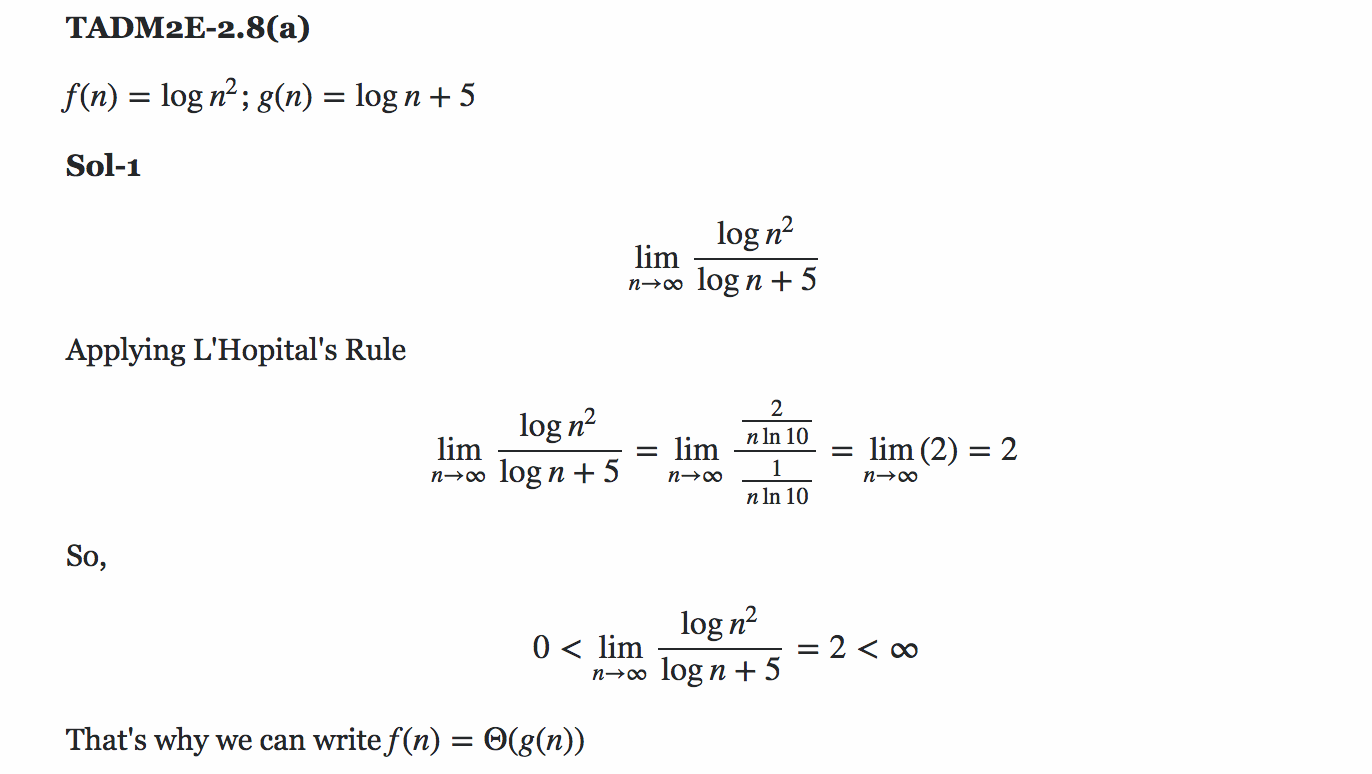
****

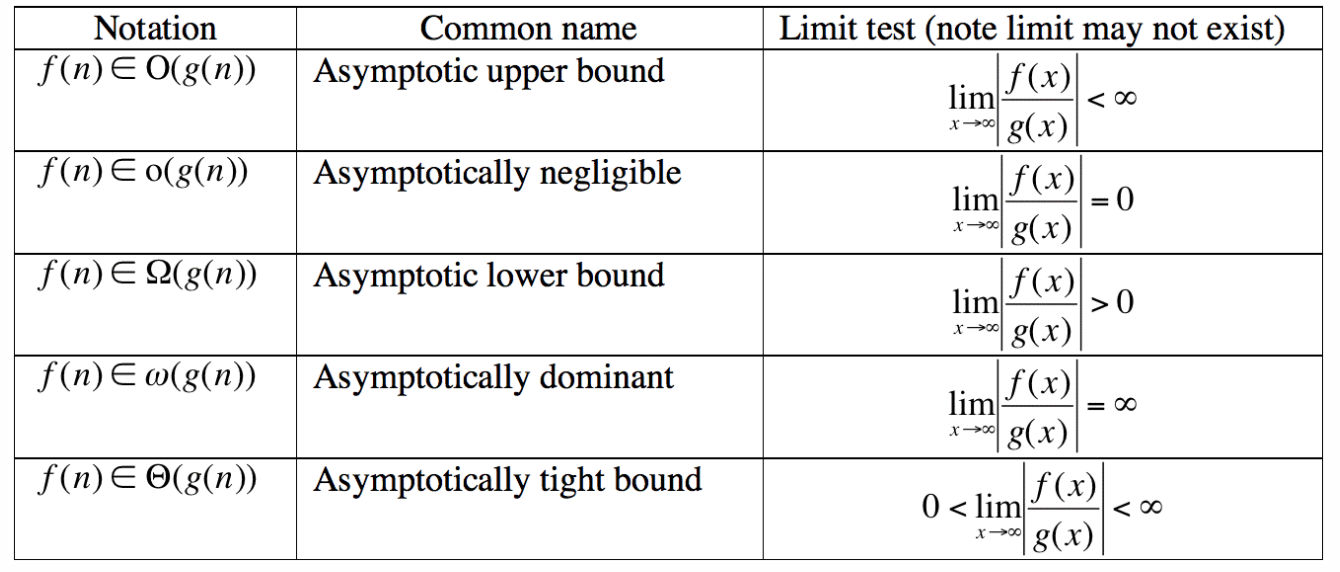
****

****

****

**f(n) = e^x g(n) = x^2**

****

****

**The following details the topics to be introduced by the review course. These topics will broadly follow this schedule.**

1. Introduction to complexity of algorithms, asymptotic notations
2. Integer arithmetic
3. Recursive and Karatsuba multiplication
4. Priority queues and heaps
5. Linear-time sorting algorithms
6. Binary search trees and average case analysis
7. AVL trees and skip-lists
8. Hashing and hash tables
9. Graphs and their representations
10. Breadth-first-search and depth-first-search
11. Strongly connected components
12. Shortest path problem
13. Dynamic programming
14. Minimum spanning trees
15. Complexity classes: P versus NP

**Chapter 1-3 Introduction to complexity of algorithms, asymptotic notations, integer arithmetic, Karatsuba (session 1,2)**

Big O-Notation

<http://bigocheatsheet.com/>

Right or wrong?

• 5n3 + 100n2 ∈ O(n3)

• n1/2 ∈ O(log n)

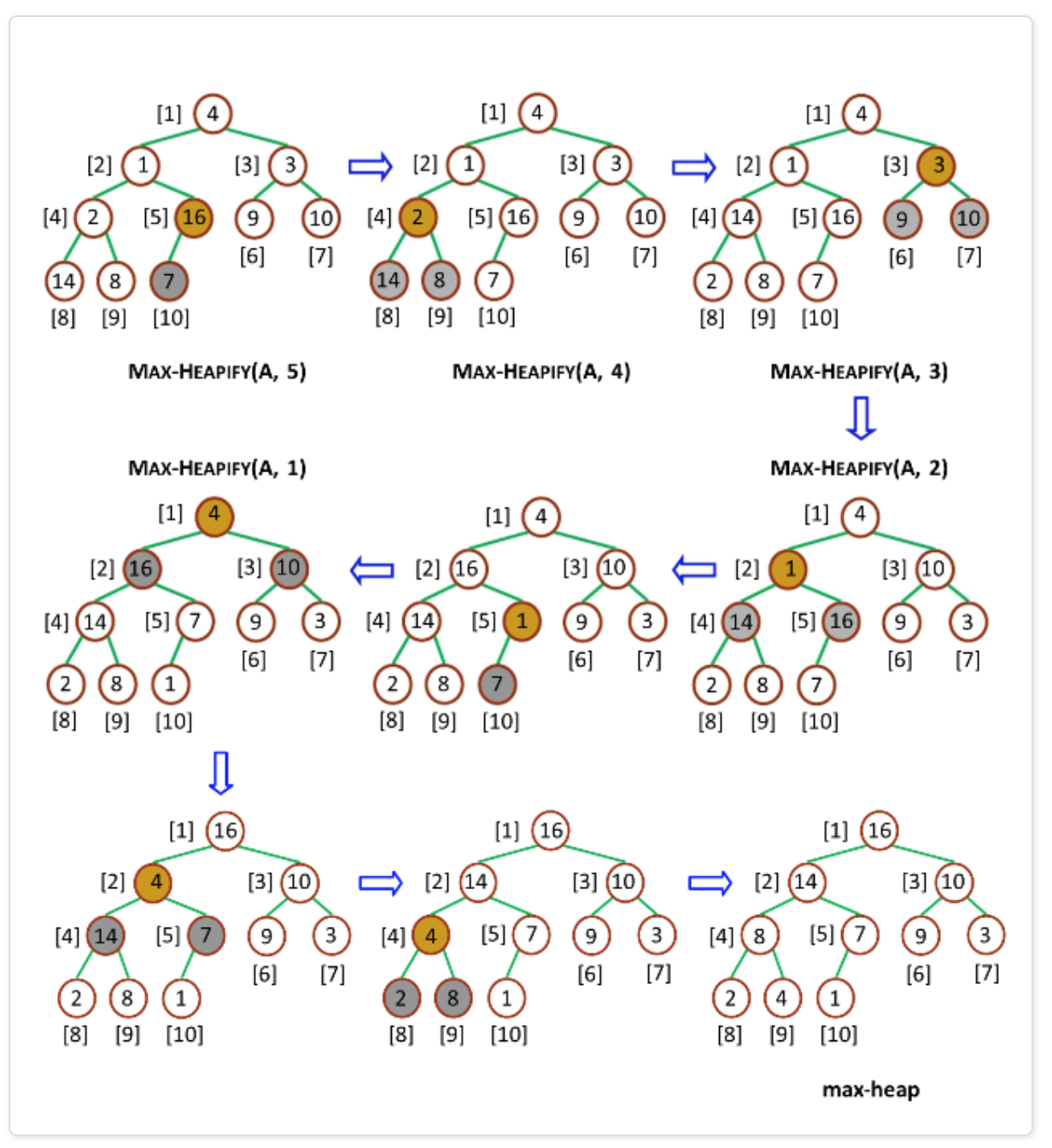
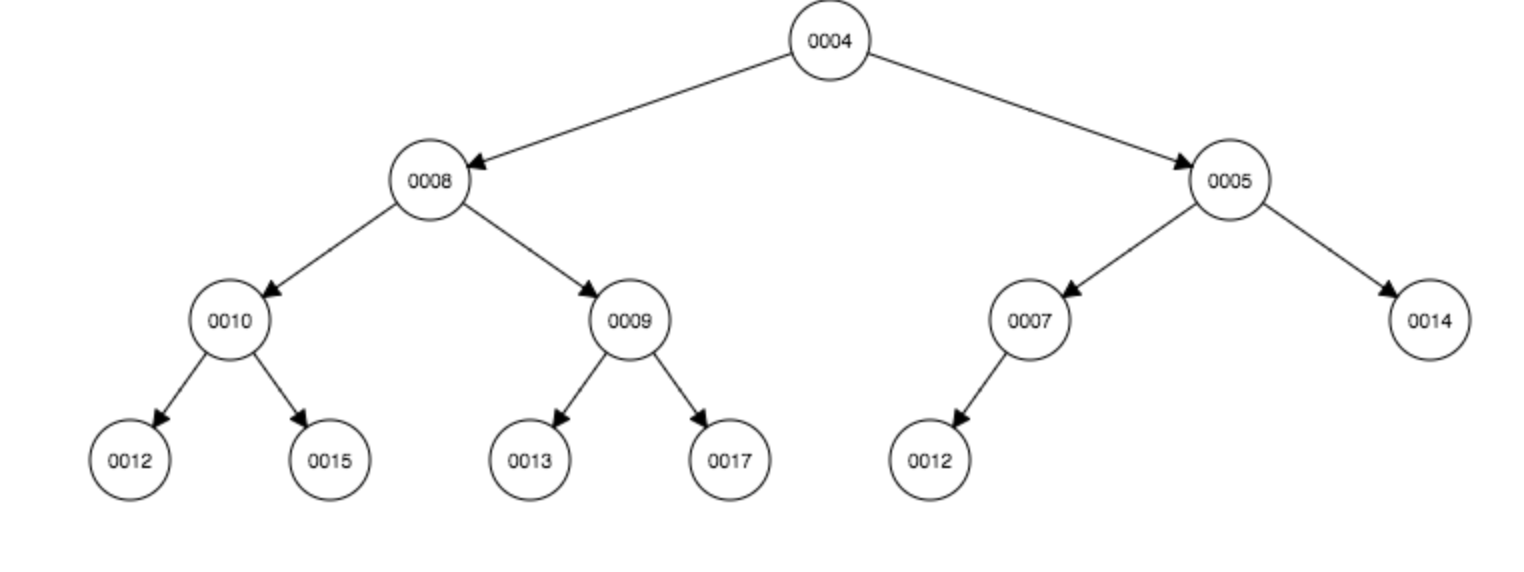
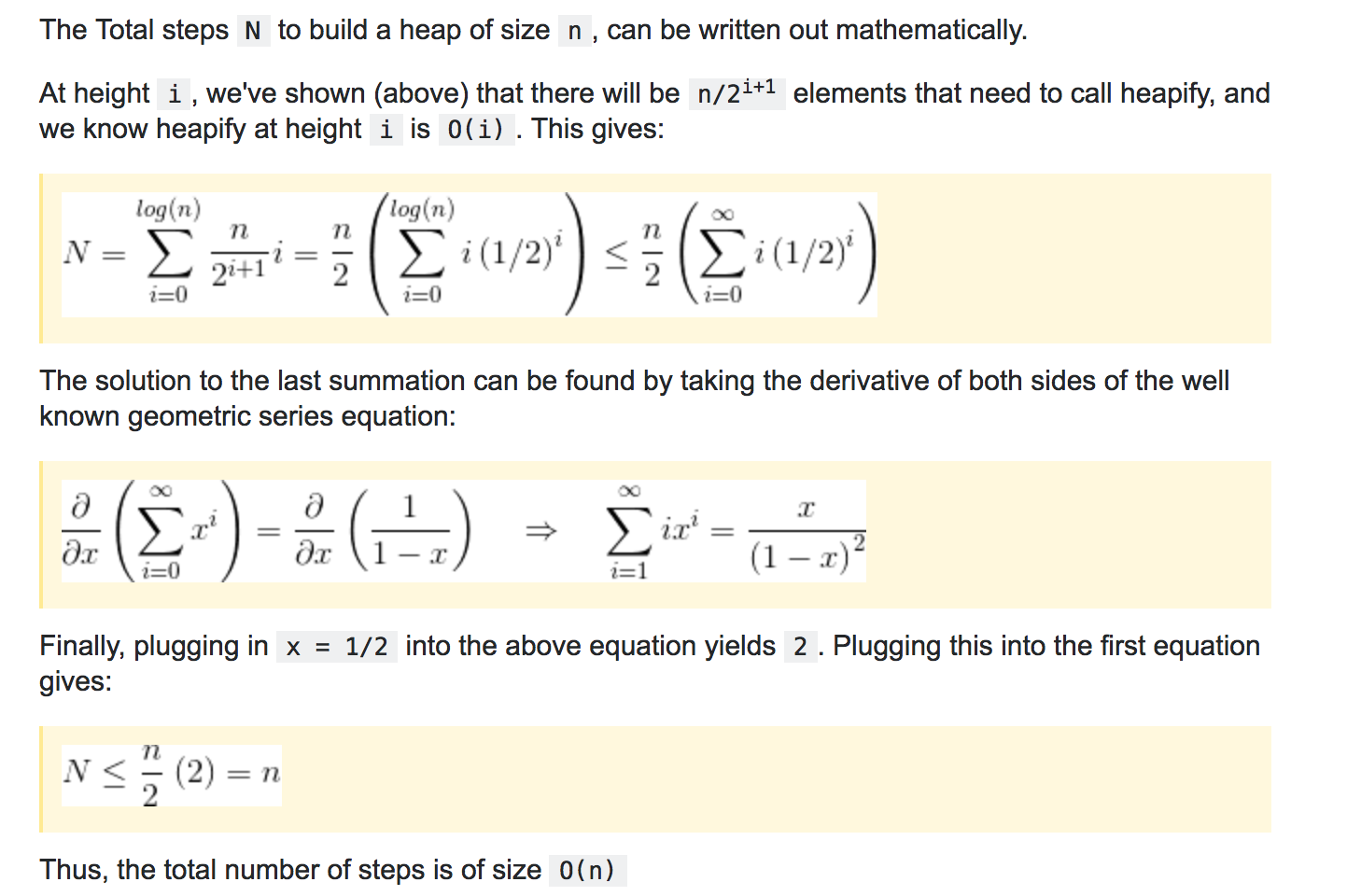
• 2n2 + 3n4 ∈ O(n2)

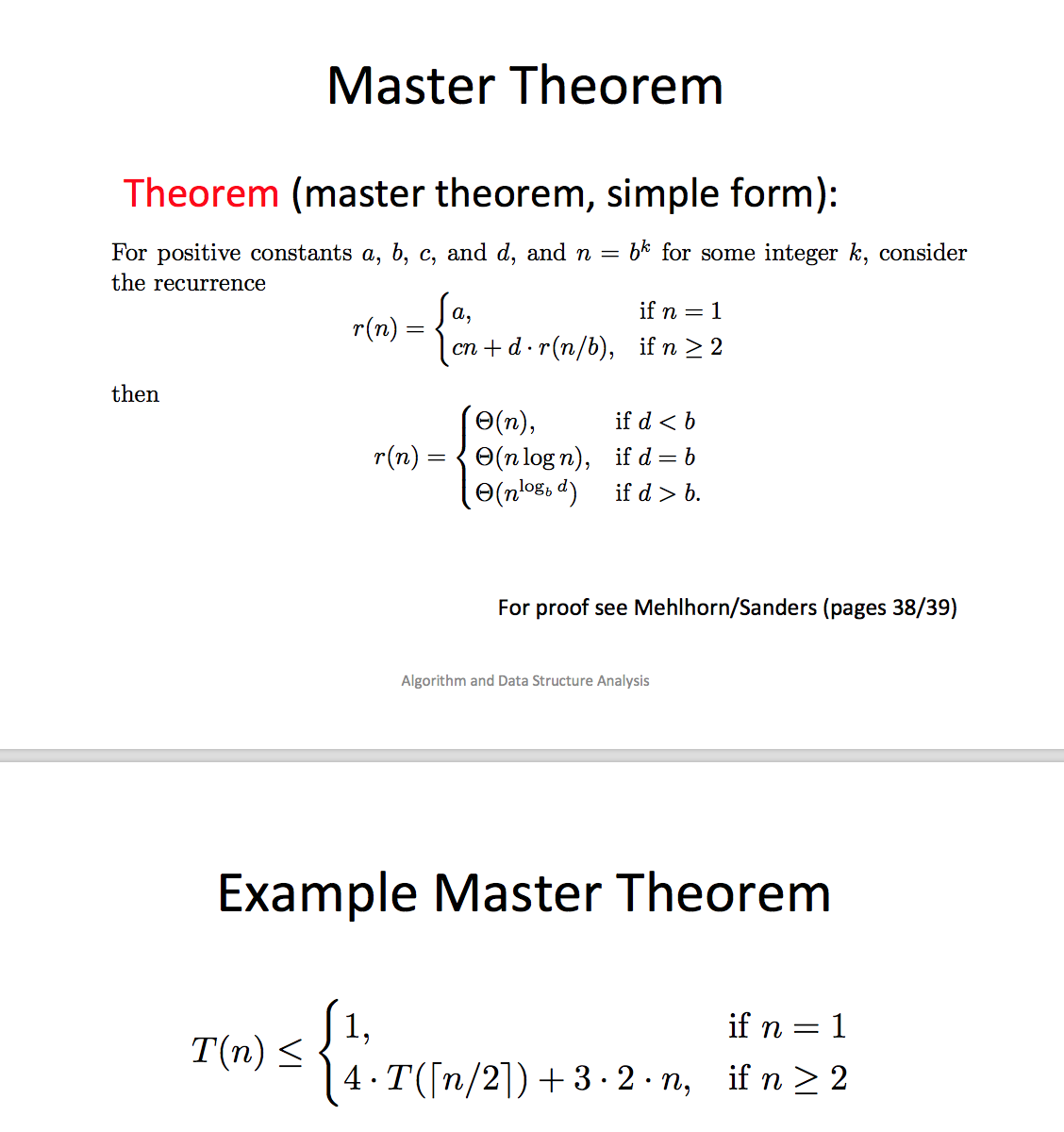
• 2n2 + 3n4 ∈ Ω(n2)

• 2n2 + 3n4 ∈ o(n5)

Integer Multiplication

1. Describe the recursive school multiplication algorithm for multiplying two n-digit numbers. Develop the recursive formula for the number of primitive operations.
2. Describe the Karatsuba multiplication algorithm for multiplying two n-digit numbers. Develop the recursive formula for the number of primitive operations. Why is the Karatsuba multiplication asymptotically faster than the school multiplication?





**Chapter 4 heaps (session 3)**

**http://www.geeksforgeeks.org/time-complexity-of-building-a-heap/**

Heaps and Heapsort

1. What are the basic properties of a heap?
2. Describe the operations insert and delete for heaps.
3. Describe the Heapsort algorithm. Analyse its runtime.

**Chapter 5 Sorting algorithms** **(session 4)**

adelaideneocs@gmail.com

Sorting algorithms

1. Merge sort
2. Heap sort
3. Quick sort
4. Radix Sort

**Chapter 6 Binary search trees and average case analysis (session 5)**

Binary Search Trees

1. What are the basic properties of binary search trees?

* The data stored at each node has a distinguished key which is unique in the tree and belongs to a *total order*. (That is, for any two non-equal keys, x,y either x < y or y < x.)
* The key of any node is greater than all keys occurring in its left subtree and less than all keys occurring in its right subtree.

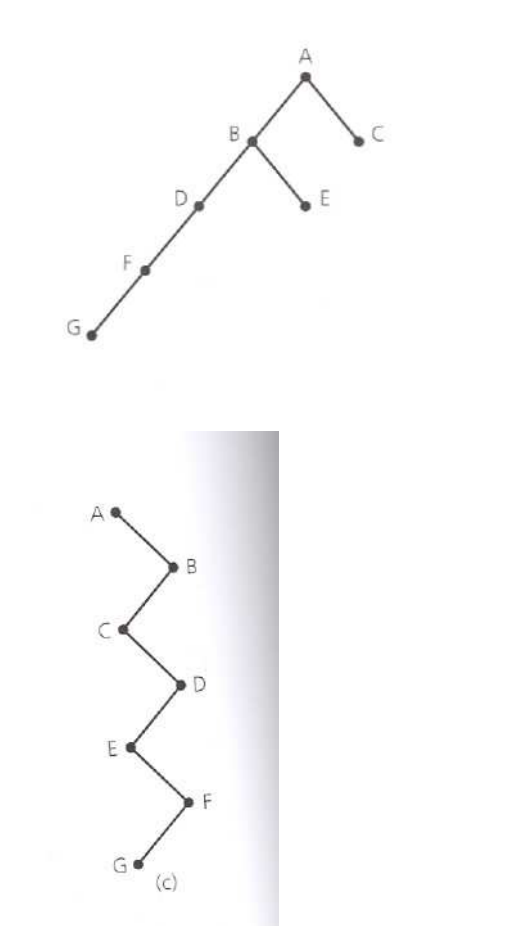
1. Describe the operations find, insert, and remove for binary search trees.
2. What is the height of a perfectly balanced binary search tree?
3. What is the worst case height of a binary search tree?
4. What is the average time for find in a binary search tree that is generated from a sequence of n randomly chosen integers?

Tree traversal

Pre- order (root, left child, right child)

In-order (left child, root, right child)

Post-order (left child, right child, root)



AVL-Trees

1. State the basic property of an AVL-tree.
2. What is the maximum height of an AVL-tree?
3. Consider an example sequence where the insertion into the AVL-tree involves all possible rotation operations.  Exercise 6 Skip Lists
4. Describe the basic properties of a Skip List.
5. What is the probability that the Skip Lists exceeds a certain height h after the insertion of n elements?
6. Show by example how find, insertion, and deleting works for Skip Lists.

Draw a sequence of diagrams showing the insertion of the values:

[9, 27, 50, 15, 2, 21, 36, 37, 51, 1 ,3, 4]

into an empty AVL tree.

You must:

首先从下至上判断第一个不平衡的节点

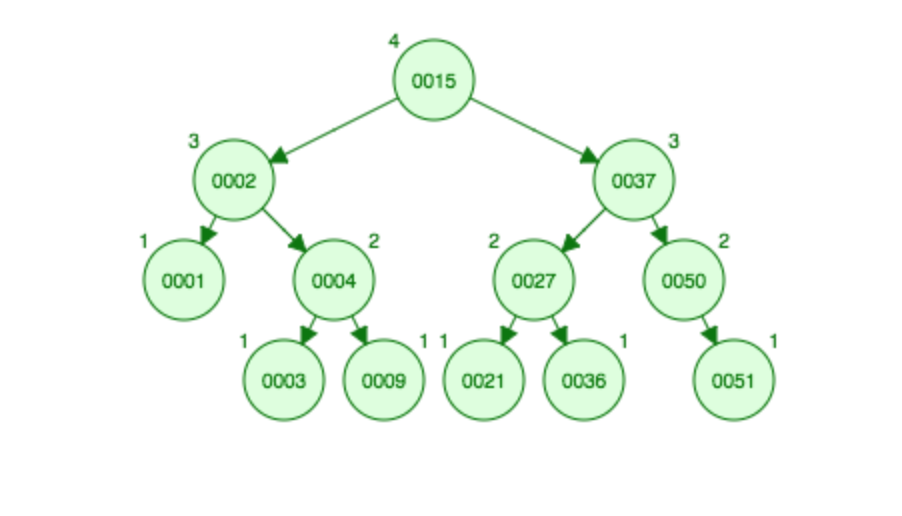
包括此节点向造成它不平衡的子树方向回溯三个节点

若此三个节点为递增或递减序列，直接下摆不平衡节点（single）

否则先把三个节点变为递增或递减序列（single）后，再下摆不平衡节点（double）

1. Show the resulting tree immediately after each insertion step (that is *before* any balancing has taken place).
2. Indicate the node(s) at which each rotation is performed.
3. Where there is a double rotation, show the tree after each single  rotation.
4. Show the resulting tree after balancing operation (s)

[7 marks]



Draw a sequence of diagrams showing the insertion of the values:

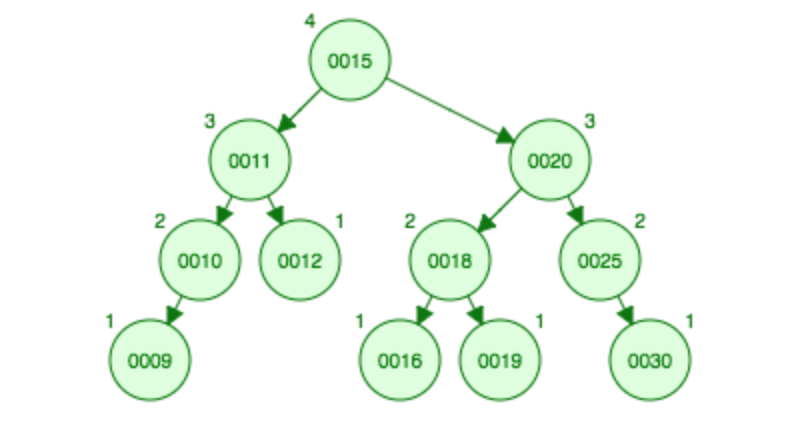
[10, 20, 15, 25, 30, 16, 18, 19, 11, 12, 9]

into an empty AVL tree.

You must:

1. Show the resulting tree immediately after each insertion step (that is *before* any balancing has taken place).
2. Indicate the node(s) at which each rotation is performed.
3. Where there is a double rotation, show the tree after each single  rotation.
4. Show the resulting tree after balancing operation (s)

[7 marks]



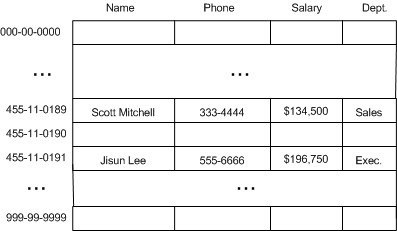
**Chapter 8 Hashing and hash tables (session 7)**

* Hashtable
  + [哈希冲突解决策略：开放寻址法（Open Addressing）](http://www.cnblogs.com/gaochundong/p/hashtable_and_perfect_hashing.html#collision_resolution_open_addressing)
    - 线性探查（Linear Probing）
    - 二次探查（Quadratic Probing）
    - 二度哈希（Rehashing）/双重哈希（Double Hashing）
  + [哈希冲突解决策略：链接技术（chaining）](http://www.cnblogs.com/gaochundong/p/hashtable_and_perfect_hashing.html#collision_resolution_chaining)

社保号的格式为 DDD-DD-DDDD（D 的范围为数字 0-9）。

如果使用 Array 存储员工信息，要查询社保号为 111-22-3333 的员工，则将会尝试遍历数组的所有位置，即执行渐进时间为 O(n) 的查询操作。好一些的办法是将社保号排序，以使[查询渐进时间降低到 O(log(n))](http://www.cnblogs.com/gaochundong/p/comparison_sorting_algorithms.html)。但理想情况下，我们更希望查询渐进时间为 O(1)。

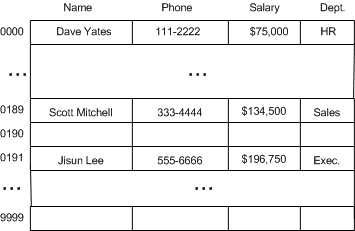
一种方案是建立一个大数组，范围从 000-00-0000 到 999-99-9999 。



这种方案的缺点是浪费空间。如果我们仅需要存储 1000 个员工的信息，那么仅利用了 0.0001% 的空间。

第二种方案就是用**哈希函数（Hash Function）**压缩序列。

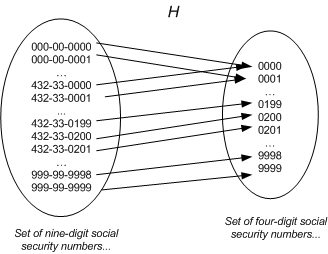
我们选择使用社保号的后四位作为索引，以减少区间的跨度。这样范围将从 0000 到 9999。



在数学上，将这种从 9 位数转换为 4 位数的方式称为**哈希转换（Hashing）**。可以将一个数组的索引空间（indexers space）压缩至相应的哈希表（Hash Table）。

在上面的例子中，哈希函数的输入为 9 位数的社保号，输出结果为后 4 位。

H(x) = last four digits of x



上图中也说明在哈希函数计算中常见的一种行为：**哈希冲突（Hash Collisions）**。即有可能两个社保号的后 4 位均为 0000。

当要添加新元素到 Hashtable 中时，哈希冲突是导致操作被破坏的一个因素。如果没有冲突发生，则元素被成功插入。如果发生了冲突，则需要判断冲突的原因。因此，**哈希冲突提高了操作的代价，Hashtable 的设计目标就是要尽可能减低冲突的发生**。

**处理哈希冲突的方式有两种：避免和解决，即冲突避免机制（Collision Avoidance）和冲突解决机制（Collision Resolution）。**

避免哈希冲突的一个方法就是选择合适的哈希函数。哈希函数中的冲突发生的几率与数据的分布有关。例如，如果社保号的后 4 位是随即分布的，则使用后 4 位数字比较合适。但如果后 4 位是以员工的出生年份来分配的，则显然出生年份不是均匀分布的，则选择后 4 位会造成大量的冲突。我们将这种选择合适的哈希函数的方法称为冲突避免机制（Collision Avoidance）。

在处理冲突时，有很多策略可以实施，这些策略称为冲突解决机制（Collision Resolution）。其中一种方法就是将要插入的元素放到另外一个块空间中，因为相同的哈希位置已经被占用。

**哈希冲突解决策略：开放寻址法（Open Addressing）**

通常采用的冲突解决策略为**开放寻址法（Open Addressing）**，将所有的元素都存放在哈希表内的数组中，不使用额外的数据结构。

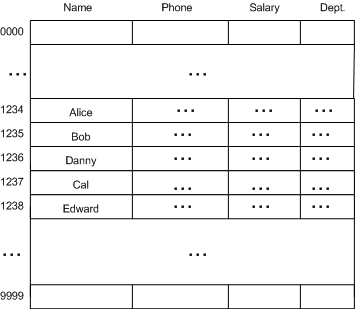
开放寻址法的最简单的一种实现就是**线性探查（Linear Probing）**，步骤如下：

1. 当插入新的元素时，使用哈希函数在哈希表中定位元素位置；
2. 检查哈希表中该位置是否已经存在元素。如果该位置内容为空，则插入并返回，否则转向步骤 3。
3. 如果该位置为 i，则检查 i+1 是否为空，如果已被占用，则检查 i+2，依此类推，直到找到一个内容为空的位置。

现在如果我们要将五个员工的信息插入到哈希表中：

* Alice (333-33-1234)
* Bob (444-44-1234)
* Cal (555-55-1237)
* Danny (000-00-1235)
* Edward (111-00-1235)

则插入后的哈希表可能如下：



元素的插入过程：

* Alice 的社保号被哈希为 1234，因此存放在位置 1234。
* Bob 的社保号被哈希为 1234，但由于位置 1234 处已经存放 Alice 的信息，则检查下一个位置 1235，1235 为空，则 Bob 的信息就被放到 1235。
* Cal 的社保号被哈希为 1237，1237 位置为空，所以 Cal 就放到 1237 处。
* Danny 的社保号被哈希为 1235，1235 已被占用，则检查 1236 位置是否为空，1236 为空，所以 Danny 就被放到 1236。
* Edward 的社保号被哈希为 1235，1235 已被占用，检查1236，也被占用，再检查1237，直到检查到 1238时，该位置为空，于是 Edward 被放到了1238 位置。

线性探查（Linear Probing）方式虽然简单，但并不是解决冲突的最好的策略，因为它会导致同类哈希的聚集（Primary Clustering）。这导致搜索哈希表时，冲突依然存在。例如上面例子中的哈希表，如果我们要访问 Edward 的信息，因为 Edward 的社保号 111-00-1235 哈希为 1235，然而我们在 1235 位置找到的是 Bob，所以再搜索 1236，找到的却是 Danny，以此类推直到找到 Edward。

一种改进的方式为**二次探查（Quadratic Probing）**，即每次检查位置空间的步长为平方倍数。也就是说，如果位置 s 被占用，则首先检查 s + 12 处，然后检查s - 12，s + 22，s - 22，s + 32 依此类推，而不是象线性探查那样以 s + 1，s + 2 ... 方式增长。尽管如此，二次探查同样也会导致同类哈希聚集问题（Secondary Clustering）。

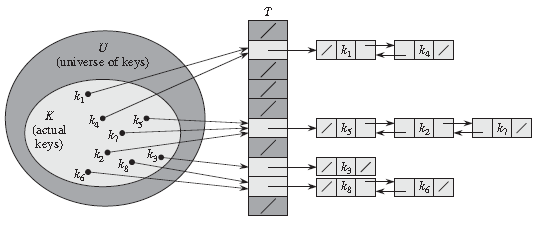
另一种改进的开放寻址法称为**二度哈希（Rehashing）**（或称为**双重哈希（Double Hashing）**）。

二度哈希的工作原理如下：

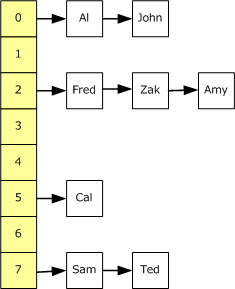
有一个包含一组哈希函数 H1...Hn 的集合。当需要从哈希表中添加或获取元素时，首先使用哈希函数 H1。如果导致冲突，则尝试使用 H2，以此类推，直到 Hn。二度哈希使用了 Θ(m2) 种探查序列，而线性探查（Linear Probing）和二次探查（Quadratic Probing）使用了Θ(m) 种探查序列，故二度哈希提供了更好的避免冲突的策略。

**哈希冲突解决策略：链接技术（chaining）**

**链接技术（chaining）**是一种冲突解决策略（Collision Resolution Strategy）。在链接法中，把哈希到同一个槽中的所有元素都放到一个链表中。



使用探查技术（probing）时，如果发生冲突，则将尝试列表中的下一个位置。如果使用二度哈希（rehashing），则将导致所有的哈希被重新计算。而**链接技术（chaining）**将采用额外的数据结构来处理冲突，其将哈希表中每个位置（slot）都映射到了一个链表。当冲突发生时，冲突的元素将被添加到桶（bucket）列表中，而每个桶都包含了一个链表以存储相同哈希的元素。



上图中的哈希表包含了 8 个桶（bucket），也就是自顶向下的黄色背景的位置。如果一个新的元素要被添加至哈希表中，将会被添加至其 Key 的哈希所对应的桶中。如果在相同位置已经有一个元素存在了，则将会将新元素添加到列表的前面。

使用链接技术添加元素的操作涉及到哈希计算和链表操作，但其仍为常量，渐进时间为 O(1)。而进行查询和删除操作时，其平均时间取决于元素的数量和桶（bucket）的数量。具体的说就是运行时间为 O(n/m)，这里 n 为元素的总数量，m 是桶的数量。但通常对哈希表的实现几乎总是使 n = O(m)，也就是说，元素的总数绝不会超过桶的总数，所以 O(n/m) 也变成了常量 O(1)。

Hashing--

Hashtable (key---mapping--- value) collision

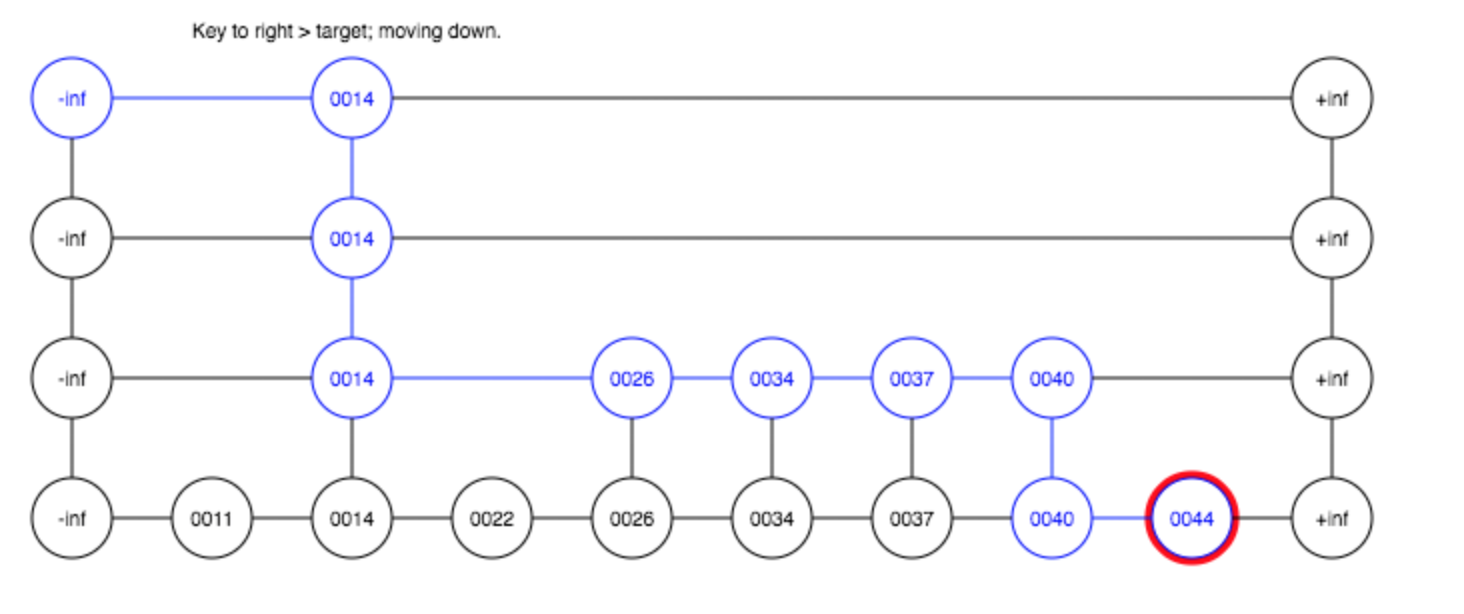
* + [哈希冲突解决策略：开放寻址法（Open Addressing）](http://www.cnblogs.com/gaochundong/p/hashtable_and_perfect_hashing.html#collision_resolution_open_addressing)
    - 线性探查（Linear Probing）
    - 二次探查（Quadratic Probing）
    - 二度哈希（Rehashing）/双重哈希（Double Hashing）
  + [哈希冲突解决策略：链接技术（chaining）](http://www.cnblogs.com/gaochundong/p/hashtable_and_perfect_hashing.html#collision_resolution_chaining)

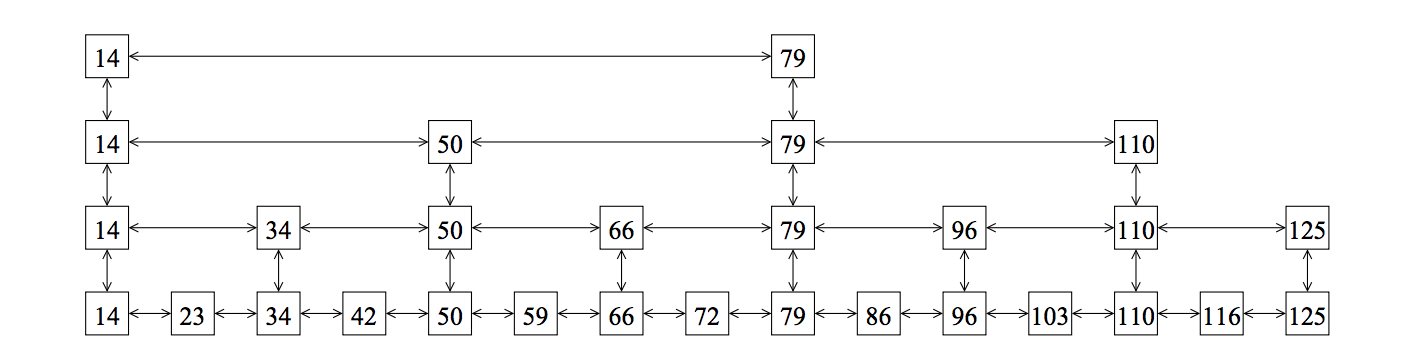
Draw the 11-item hash table resulting from hashing the keys 12,44,13,88,23,94,11,39,20,16,5 using the hash function h(x) = (2x+5) mod 11 and handling collisions using chaining and linear probing.

Given the values {2341, 4234, 2839, 430, 22, 397, 3920}, a hash table of size 7, and hash function h(x) = x mod 7, show the resulting tables after inserting the values in the given order with each of these collision strategies： chaining and linear probing and quadratic probing and

double hashing with second hash function h'(x) = (2x - 1) mod 7.

**Chapter 7 AVL trees and skip list (session 6)**



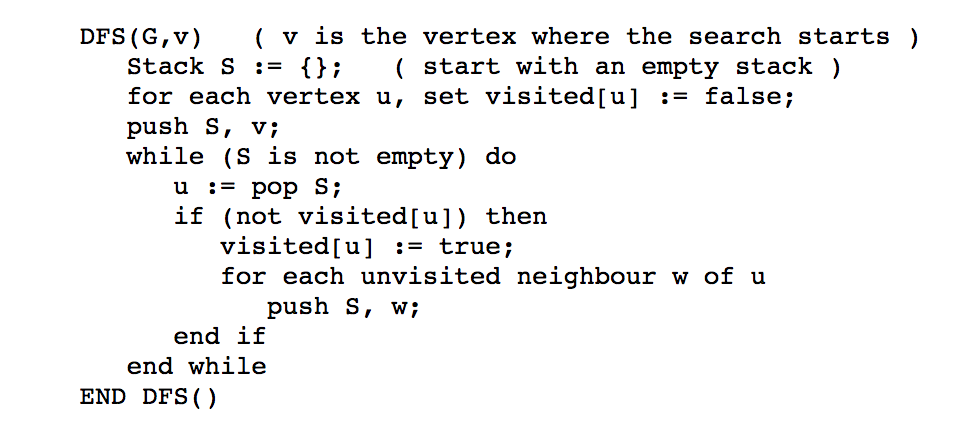


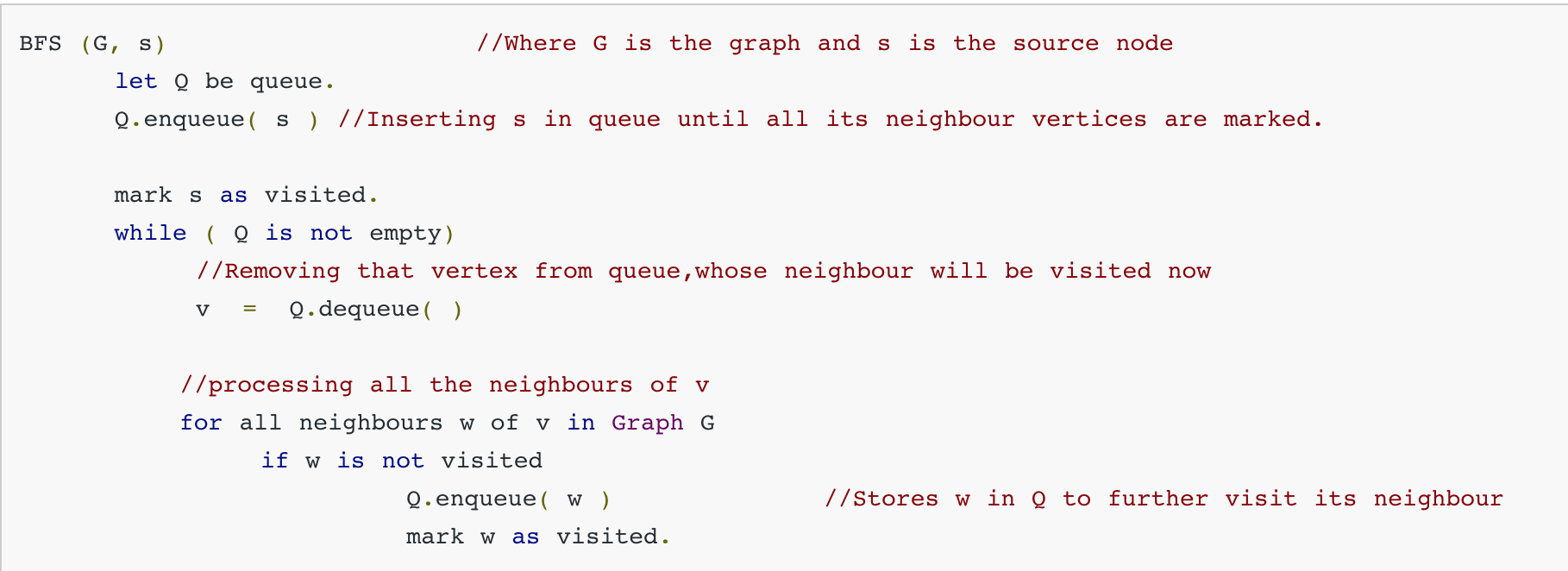
**Chapter 9-11 Graphs and their representations (DFS/BFS) & SCC**

**(session 8, 9)**

Graph Algorithms

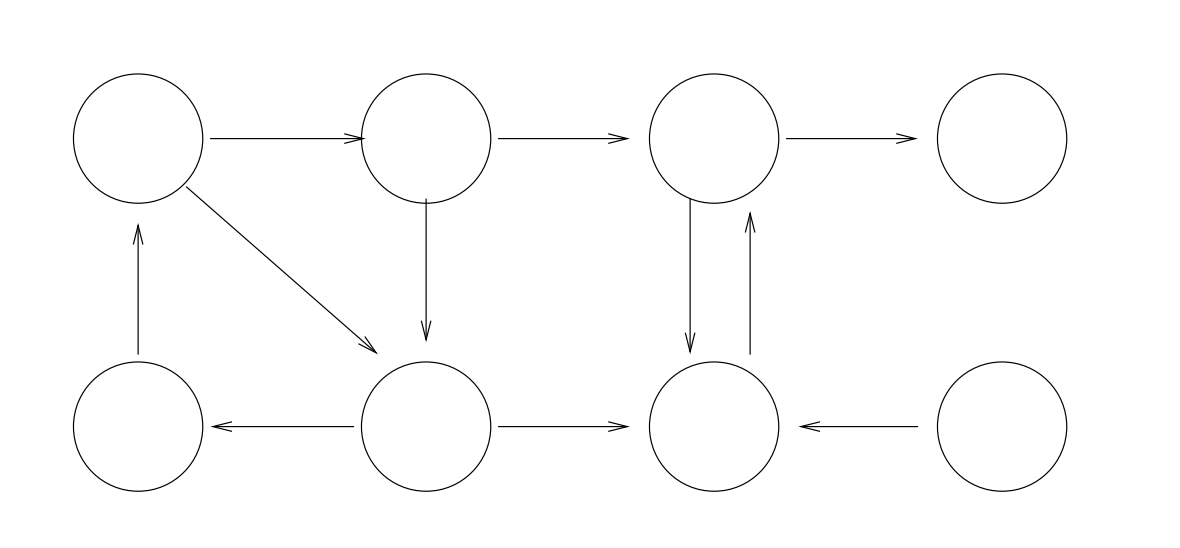
Give the pseudo code of DFS and BFS and show how they work on an example graph.



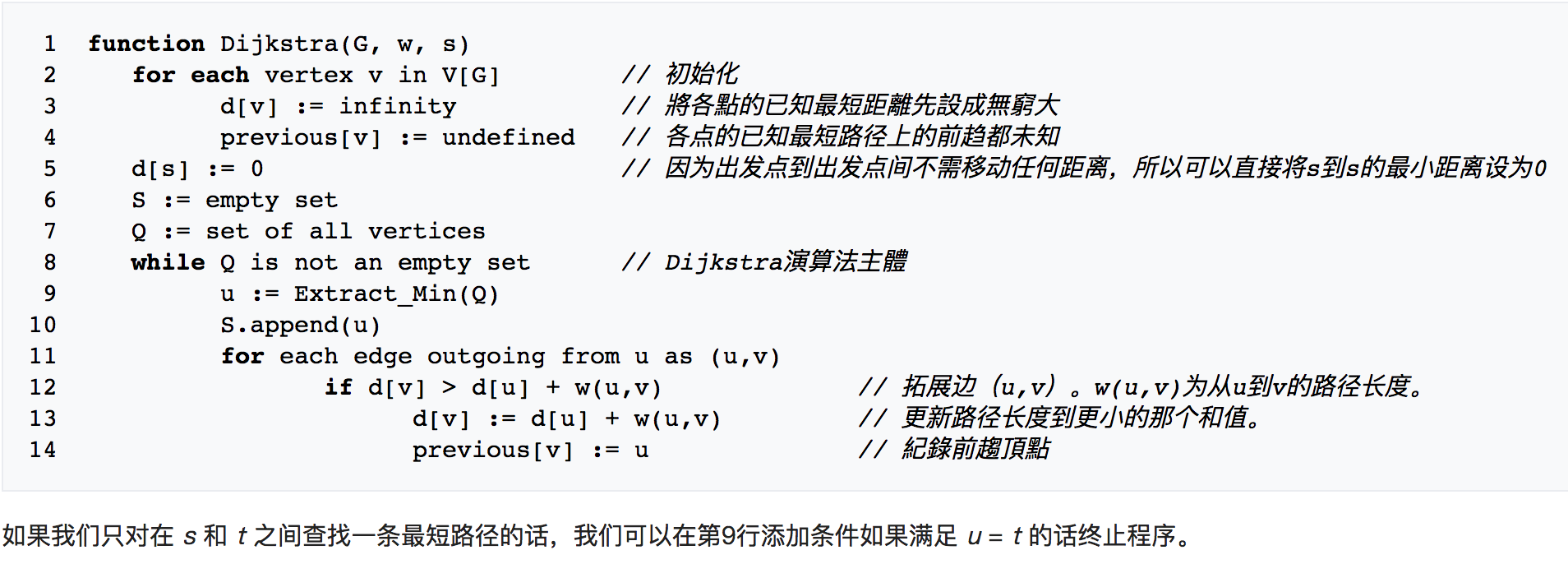


topological sort, ppt

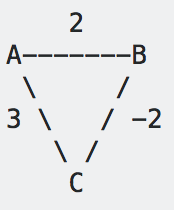
scc example

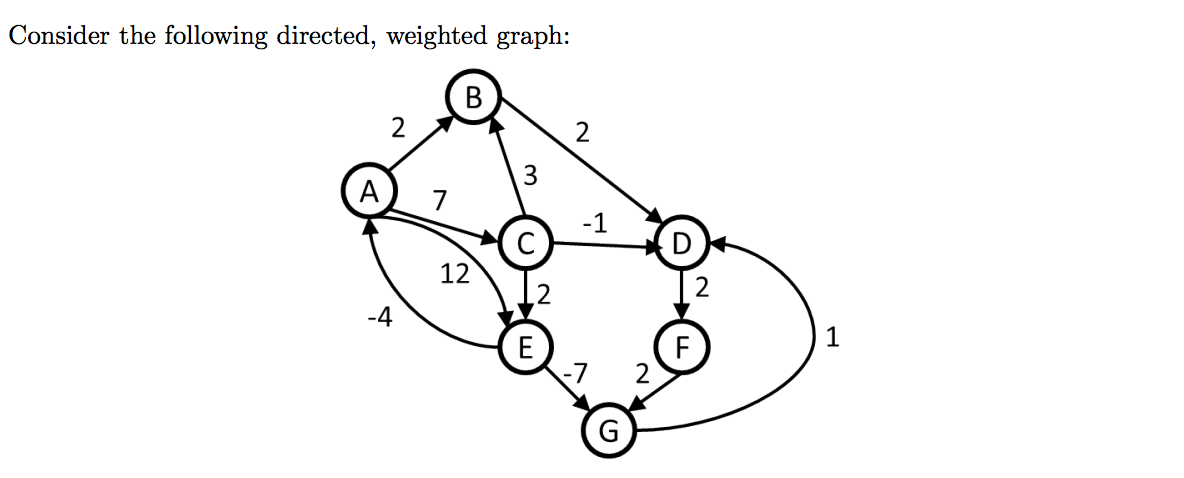


Dijkstra



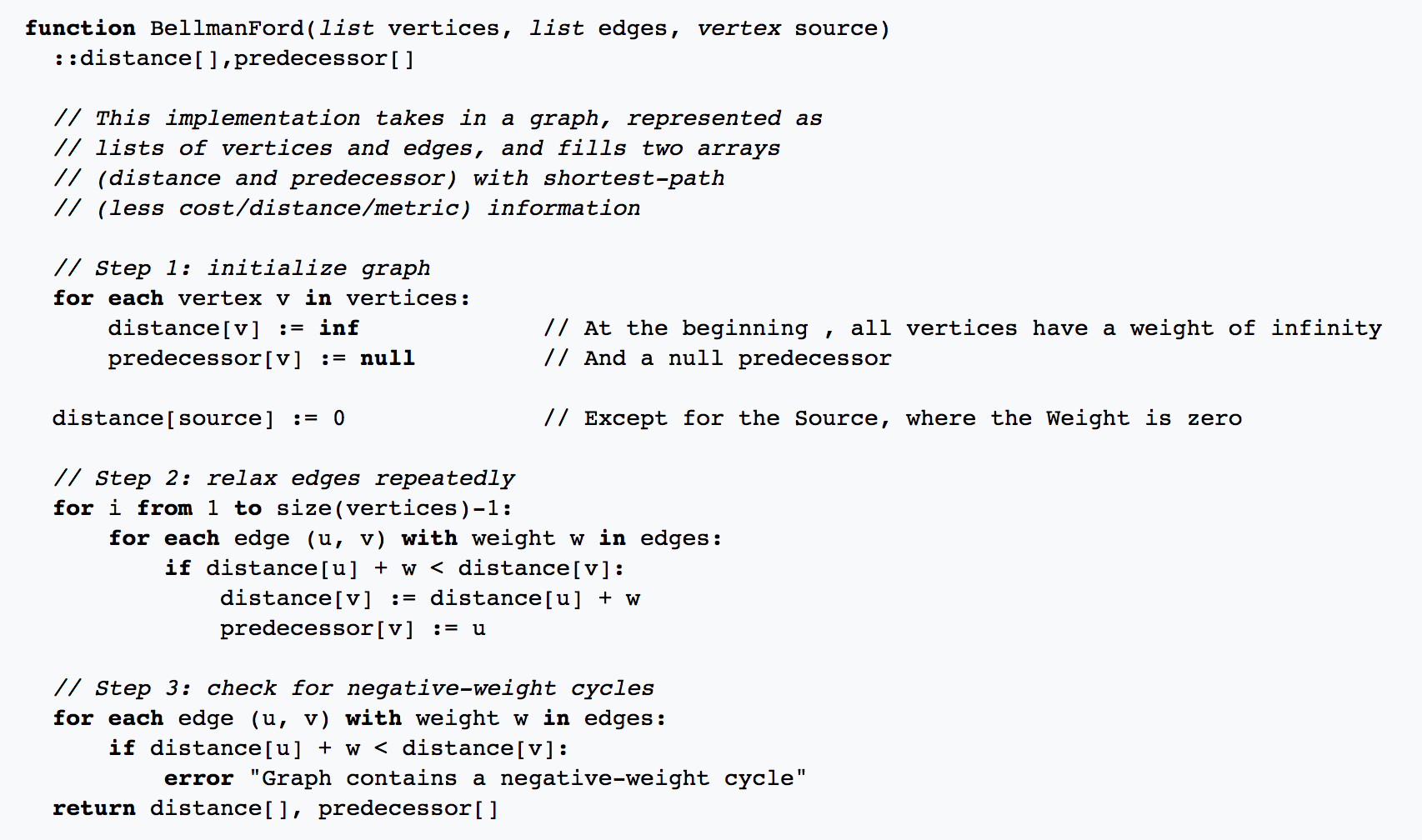
O(v^2)



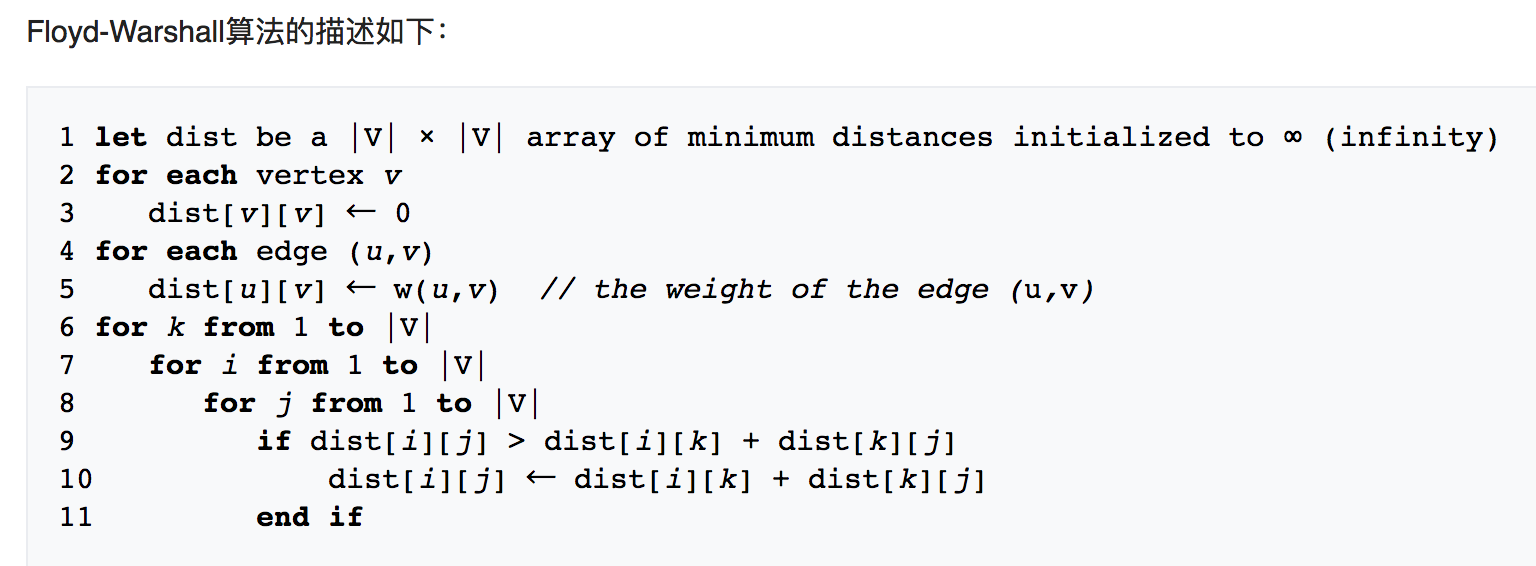


As long as the graph does not contain a negative cycle (a directed cycle whose edge weights have a negative sum), it will have a shortest path between any two points, but Dijkstra's algorithm is not designed to find them. The best-known algorithm for finding single-source shortest paths in a directed graph with negative edge weights is the [Bellman-Ford algorithm](http://en.wikipedia.org/wiki/Bellman-Ford_algorithm). This comes at a cost, however: Bellman-Ford requires O(|V|·|E|) time, while [Dijkstra's](http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) requires O(v^2)

Bellman-ford O(V\*E)

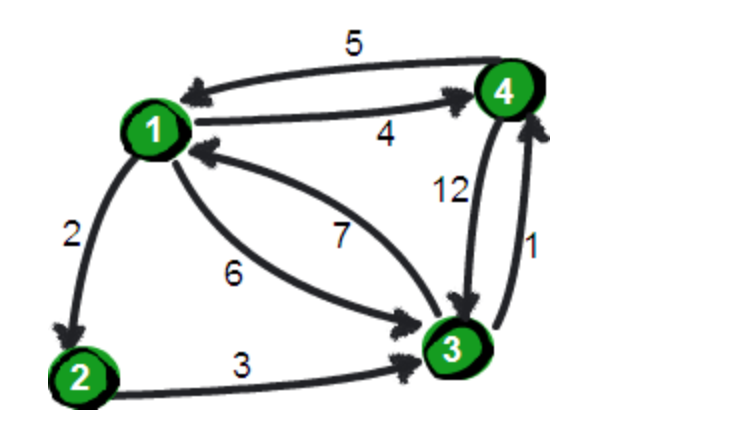


1. Using Bellman Ford to fill in the iteration form starting from vertex E
2. Show the final shortest paths from vertex E with corresponding distances
3. Check the relaxation condition one additional time for each edge. If there exists a negative weight cycle in the graph, mark it below.



O(v^3)

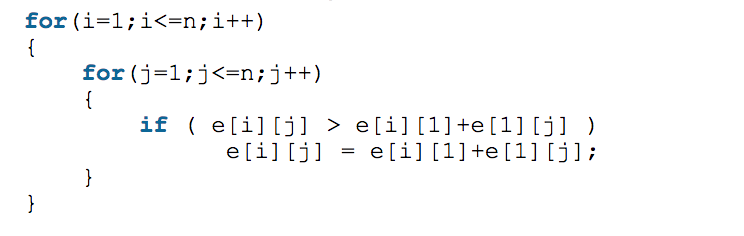
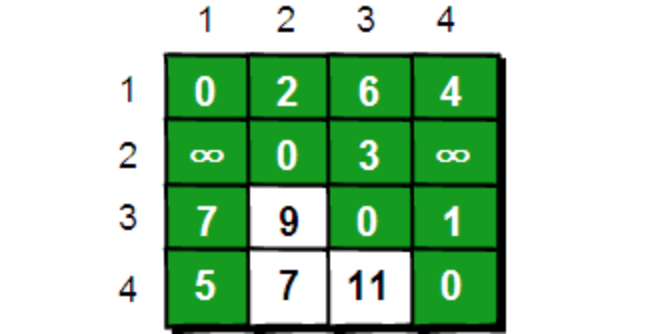




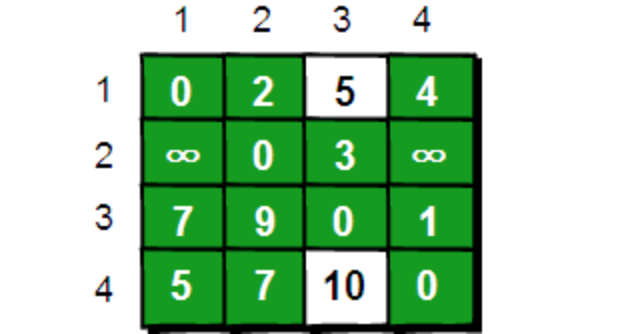
# **Adjacency matrix**



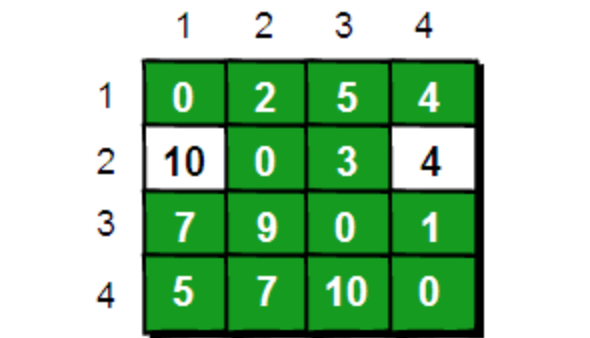
Only pass city 1

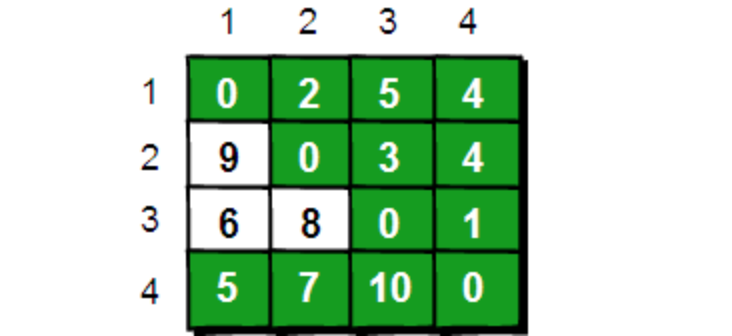
pass city 1 and 2



pass city 1, 2 and 3



pass call the cities



Give an algorithm that computes the strongly connected components for a given directed graph. What is the running time?

**Chapter 12 shortest path problems (session 10)**

For a general weighted graph, we can calculate single source shortest distances in O(VE) time using [Bellman–Ford Algorithm](http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/). For a graph with no negative weights, we can do better and calculate single source shortest distances in O(E + VLogV) time using [Dijkstra’s algorithm](http://www.geeksforgeeks.org/greedy-algorithms-set-7-dijkstras-algorithm-for-adjacency-list-representation/)

Shortest Paths

1. Give an algorithm that solves the single-source-shortest path problem for a given weighted graph where the edge weights are positive.
2. Show the execution of this algorithm on an example graph.
3. Give the Bellman-Ford algorithm for solving the all-pairs-shortest-path problem with positive edge weights.

**Chapter 13 Dynamic programming (not released)**

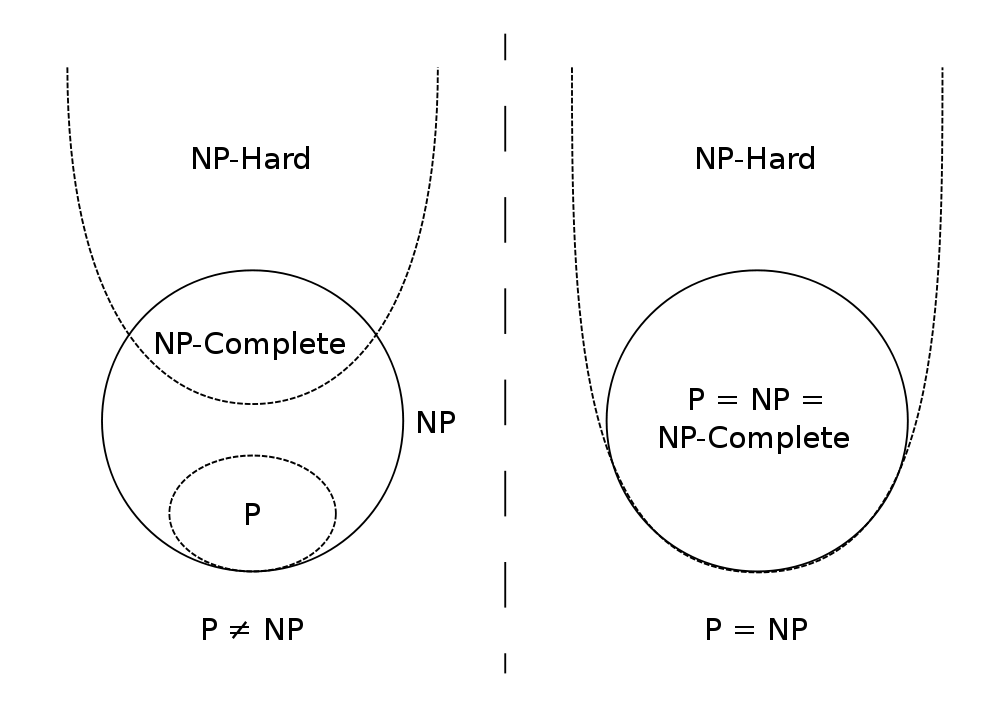
**Chapter 14 Minimum spanning trees (session 11)**

For a graph with **V** vertices **E** edges, Kruskal's algorithm runs in **O(E log V)** time and Prim's algorithm can run in **O(E + V log V)** amortized time, if you use a [Fibonacci Heap](http://en.wikipedia.org/wiki/Fibonacci_heap).

Minimum Spanning Trees

1. Give Kruskal’s algorithm for the computation of a minimum spanning tree and show the execution of this algorithm on an example graph.
2. Analyse the runtime of Kruskal’s algorithm and show how union and find operations can be supported efficiently.
3. Give the Prim algorithm for the computation of a minimum spanning tree and show the execution of this algorithm on an example graph.

**Chapter 15 Complexity classes: P versus NP (session 12)**

****

**P类问题的概念：如果一个问题可以找到一个能在多项式的时间里解决它的算法。**

**NP问题不是非P类问题。NP问题是指可以在多项式的时间里验证一个解的问题。通常只有NP问题才可能找到多项式的算法。所有的P类问题都是NP问题。人们普遍认为，P=NP不成立。**

**NPC问题的定义非常简单。同时满足下面两个条件的问题就是NPC问题。首先，它得是一个NP问题；然后，所有的NP问题都可以约化到它。**

**NP-Hard问题是这样一种问题，它满足NPC问题定义的第二条但不一定要满足第一条。**

**证明一个问题是NP-hard的。对于证明一个问题是NP-hard，我们经常用到的一个technique是归约（reduction），通常用<=这个符号来表示，如P<=Q，这个就表示P is reducible to Q or Q is the reduction from P or P is reduced to Q. P问题可以归约到Q问题。可以把P归约到Q**

**P: 如果一个问题可以找到一个能在多项式的时间里解决它的算法，那么这个问题就属于P问题**

**NP: NP问题不是非P类问题。NP问题是指可以在多项式的时间里验证一个解的问题。NP问题的另一个定义是，可以在多项式的时间里猜出一个解的问题。**

P and NP

1. Characterize the classes P and NP.
2. What does it mean that a problem is NP-hard?
3. How do you prove that a problem is NP-complete?
4. Show that the Traveling Salesman Problem is NP-complete. You can assume that the Hamiltonian Cycle problem is NP-complete.
5. Give three other examples of NP-complete problems.

****