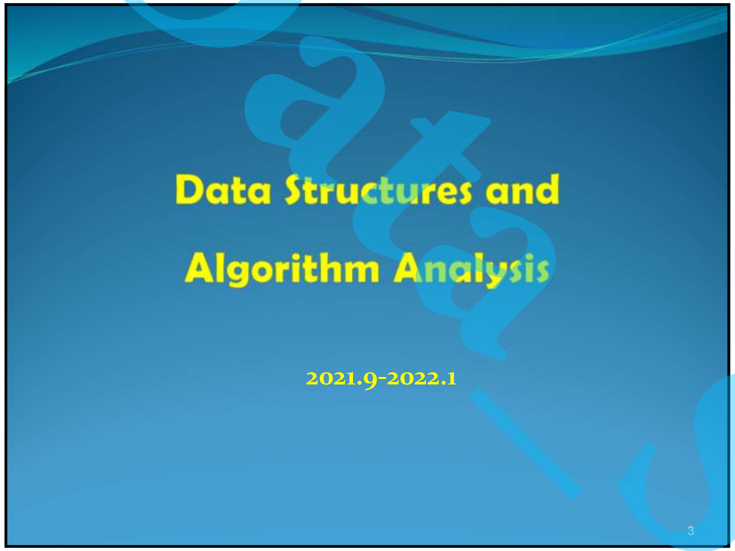
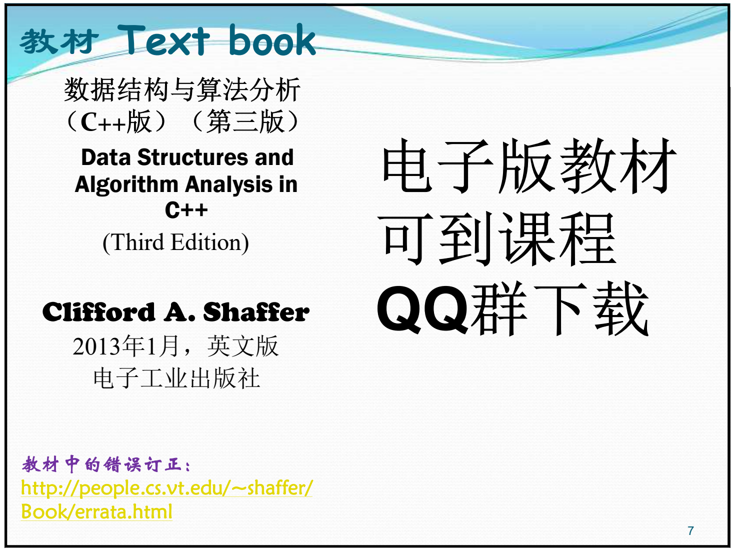


Data Structures and Algorithm Analysis

2021.9-2022.1



教材 Text book

数据结构与算法分析
（C++版）（第三版）

Data Structures and Algorithm Analysis in C++

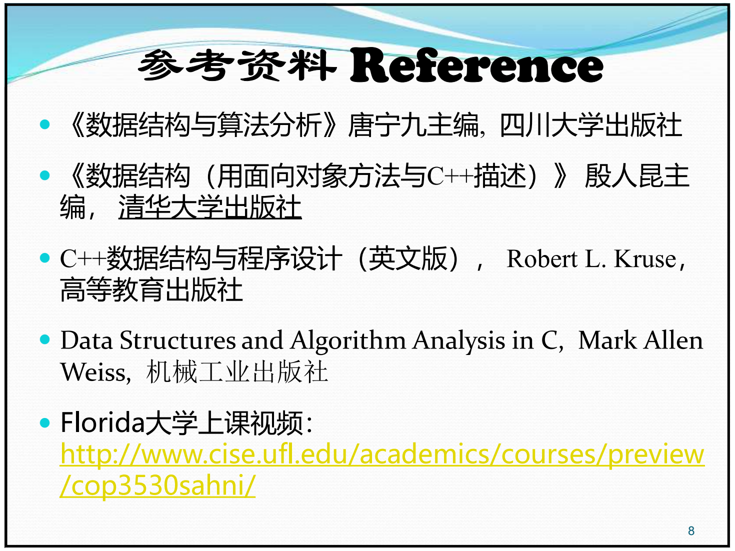
(Third Edition)

Clifford A. Shaffer

2013年1月，英文版
电子工业出版社

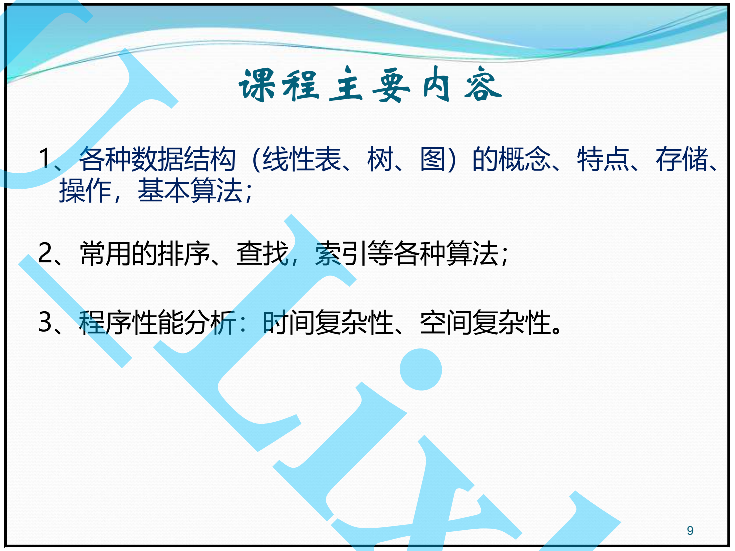
电子版教材可到课程QQ群下载

教材中的错误订正：
http://people.cs.vt.edu/~shaffer/Book/errata.html



参考资料 Reference

- 《数据结构与算法分析》唐宁九主编，四川大学出版社
- 《数据结构（用面向对象方法与C++描述）》 殷人昆主编， 清华大学出版社
- C++数据结构与程序设计（英文版）， Robert L. Kruse，高等教育出版社
- Data Structures and Algorithm Analysis in C，Mark Allen Weiss，机械工业出版社
- Florida大学上课视频：http://www.cise.ufl.edu/academics/courses/preview/cop3530sahni/



课程主要内容

1、各种数据结构（线性表、树、图）的概念、特点、存储、操作，基本算法；

2、常用的排序、查找，索引等各种算法；

3、程序性能分析：时间复杂性、空间复杂性。

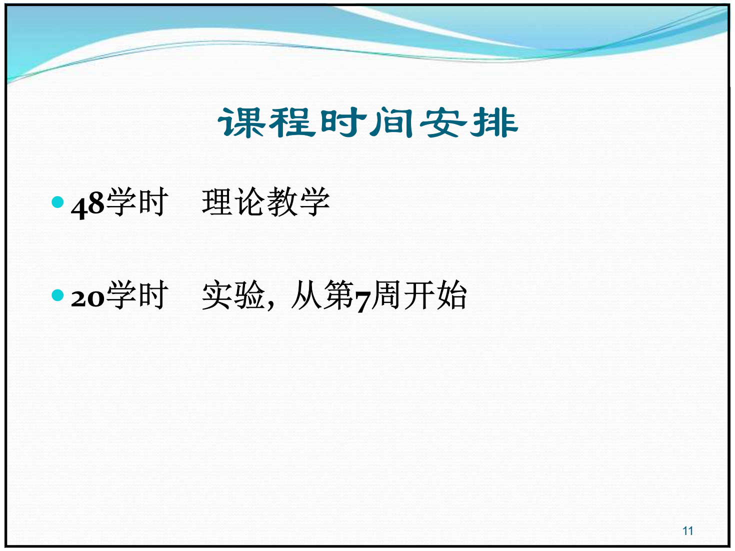
前导课程：

📖离散数学；

- 具备一定的离散数学知识（集合，关系，对数等）和一定数学证明方法。

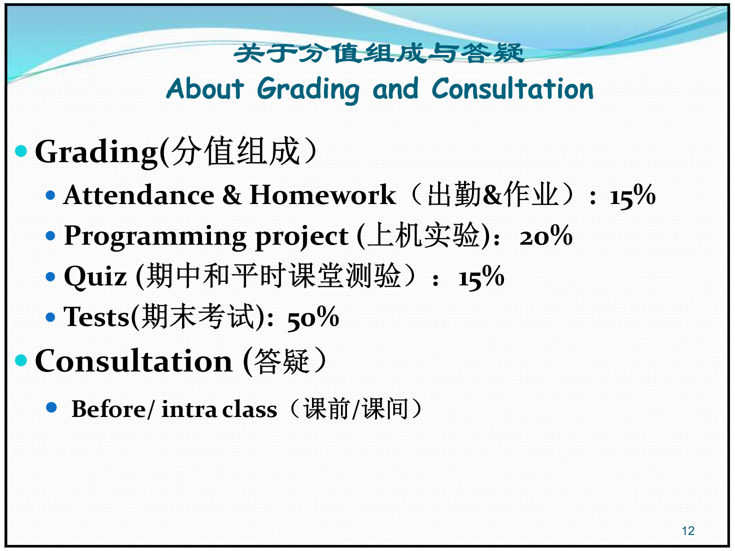
📖C++或C程序设计

- 具备C语言、面向对象程序设计语言知识



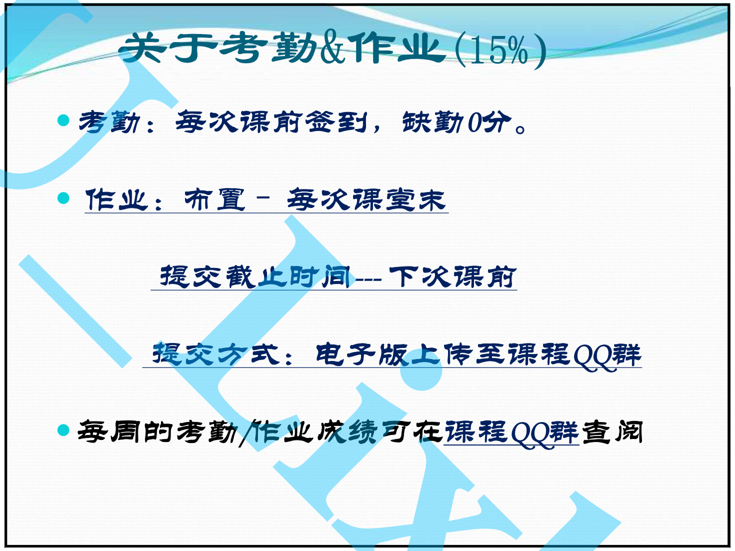
课程时间安排

- 48学时　理论教学
- 20学时　实验，从第7周开始



关于分值组成与答疑
About Grading and Consultation

- Grading(分值组成）
  - Attendance & Homework（出勤&作业）：15%
  - Programming project (上机实验)：20%
  - Quiz (期中和平时课堂测验）：15%
  - Tests(期末考试)：50%
- Consultation (答疑）
  - Before/ intra class（课前/课间）



关于考勤&作业(15%)

- 考勤：每次课前签到，缺勤0分。
- 作业：布置 – 每次课堂末

  提交截止时间--- 下次课前

  提交方式：电子版上传至课程QQ群
- 每周的考勤/作业成绩可在课程QQ群查阅

关于实验(20%)

- 共2个题目(具体内容实验时布置)，以小组为单位完成。
- 对于每个实验，会给出程序验收及报告提交的最后期限（请认真关注），在规定的提交时间后两周内完成评分，并会上传成绩至课程网站
- 每个同学可上网查看自己的实验得分，如果你对分数有异议，请在一周内跟老师提出复查申请。

关于延迟提交

- 作业没按期提交一律 0 分。
  - 请不要找老师给予特别照顾，不接受任何借口。
- 实验延迟提交：
  - 根据延期时间扣分，布置实验时会进行详细说明

关于小测验和期末考试

- 期中考试与平时小测验 15%
  - 期中考试大约安排在第9-12周，会提前一周通知
  - 平时课堂测验不定期进行，缺考0分
- 期末考试 50%
  - 由学院统一安排，学院网站上发布通知，大约在18~20周。
  - **最低线（生死线）： *40* 分**

课程目标

- **掌握算法性能分析方法。**
- **熟练掌握各种典型数据结构的特点，存储表示，深刻了解相应算法及其实现过程；**
- **熟练掌握查找和排序的基本算法。**
- **能根据实际问题的要求设计选择合适的数据结构，具有一定的比较和选用数据结构及算法的能力。**

课件中底色说明

- 白色： 基本要求
  - 针对全体同学
  - 主要涉及需要掌握的理论知识
  - 通过实例详细讲解
- 浅绿色：高级要求
  - 各种跟编程有关的具体代码
  - 对个人动手能力有高要求的同学最好深入理解此部分
  - 不喜编程同学了解其大致框架即可

Course outline 目录

1. Introduction 绪论
2. Algorithm Analysis 算法分析方法
3. list， stack and queue 线性表，栈和队
4. Tree and **Binary Trees** 树和二叉树
5. **Internal Sorting** 内部排序
6. File processing and external sorting 文件管理&外部排序
7. **Searching** 查找
8. **Indexing** 索引
9. Graph 图

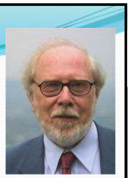
Chapter 1 Introduction

1.1 why do we study data structures

1.2 some Basic concepts in this course

1.3 Mathematical Background

1.1 why do we study data structures



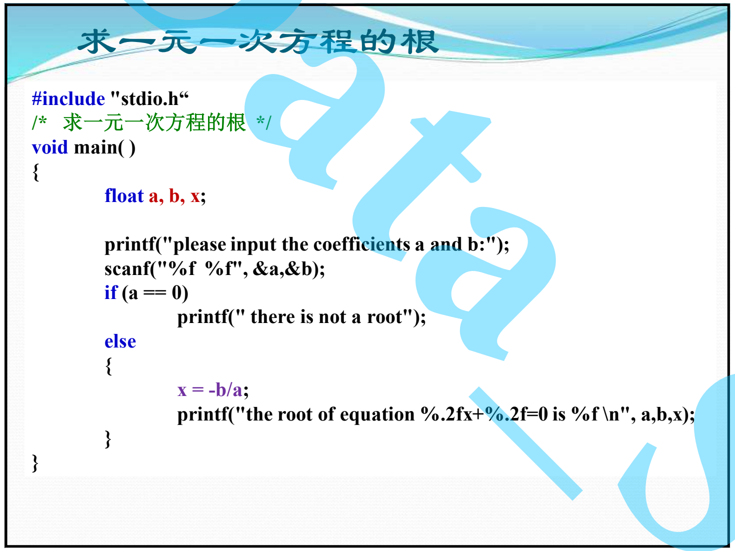
Niklaus Wirth

Programs = Algorithm + Data Structures

程序设计：为计算机处理问题
编制一组指令集

算 法：处理问题的策略

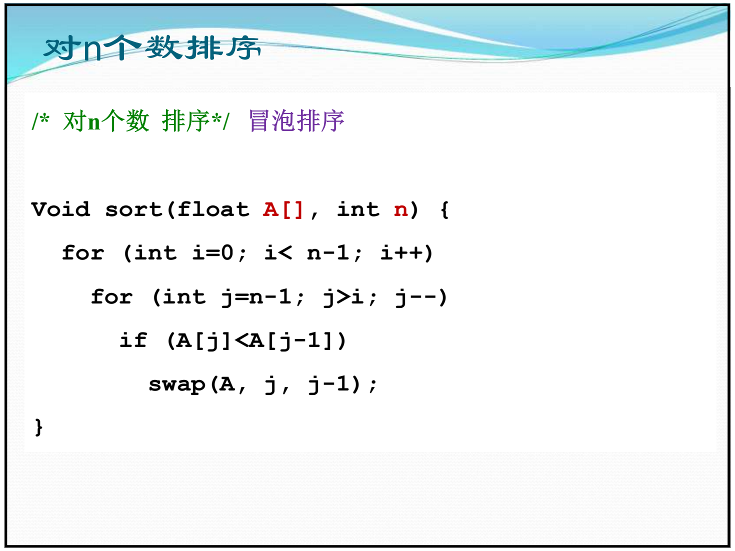
数据结构：问题中所涉及数据(集)的组织方式



求一元一次方程的根

```
#include "stdio.h“
/*  求一元一次方程的根  */
void main( )
{
         float a, b, x;

         printf("please input the coefficients a and b:");
         scanf("%f  %f", &a,&b);
         if (a == 0)
                  printf(" there is not a root");
         else
         {
                  x = -b/a;
                  printf("the root of equation %.2fx+%.2f=0 is %f \n", a,b,x);
         }
}
```



对n个数排序

/* 对n个数 排序*/ 冒泡排序

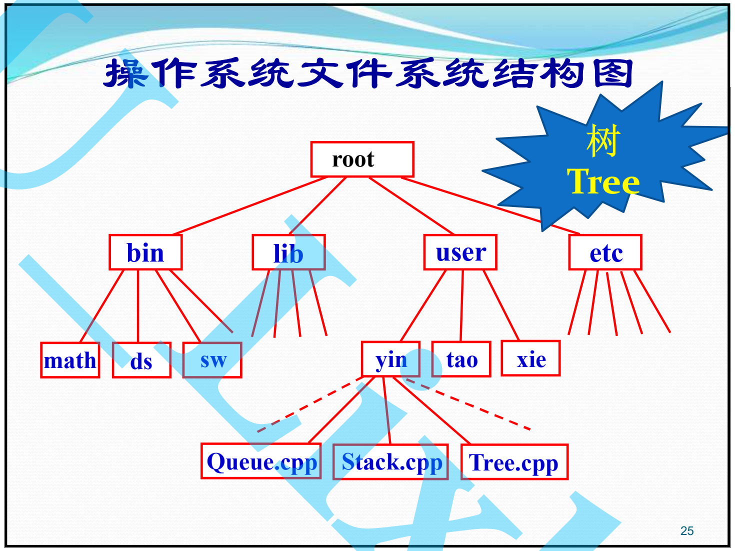
```
Void sort(float A[], int n) {
  for (int i=0; i< n-1; i++)
    for (int j=n-1; j>i; j--)
      if (A[j]<A[j-1])
        swap(A, j, j-1);
}
```

学生选课系统

线性表
List

“学生”表格

	学 号	姓　名	性别	籍　贯	出生年月
1	98131	刘激扬	男	北　京	1979.12
2	98164	衣春生	男	青　岛	1979.07
3	98165	卢声凯	男	天　津	1981.02
4	98182	袁秋慧	女	广　州	1980.10
5	98224	洪　伟	男	太　原	1981.01
6	98236	熊南燕	女	苏　州	1980.03
7	98297	宫　力	男	北　京	1981.01
8	98310	蔡晓莉	女	昆　明	1981.02
9	98318	陈　健	男	杭　州	1979.12



操作系统文件系统结构图

树
Tree

地图及图中多地点之间的路径

The Need for Data Structures

More powerful computers

⇒more complex applications.

⇒more calculations.

⇒more data

Data structures organize data

⇒ more **efficient(高效)** programs.

The choice of data structure and algorithm can make the difference between a program running in **a few seconds** or **many days**.

1.2 Some basic concepts in this course

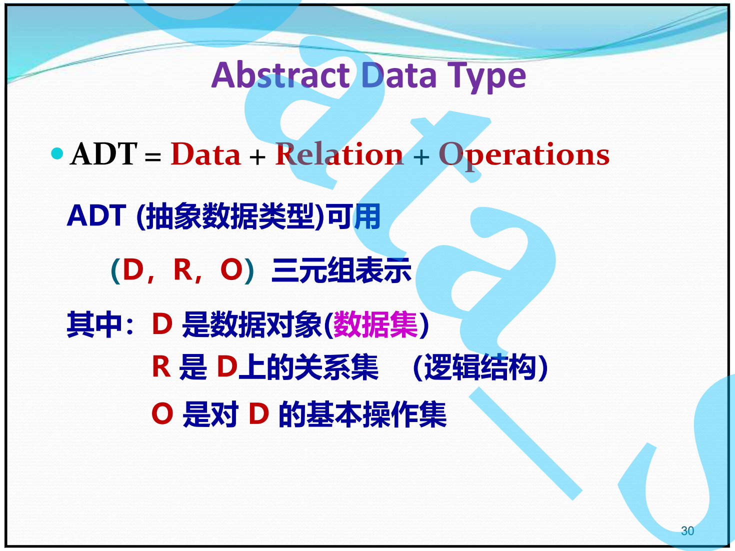
1.2.1 Abstract Data Types(抽象数据类型) and data structures(数据结构)

1.2.2 Logical(逻辑结构) vs. Physical Form(物理结构)

1.2.3 Algorithm and Program

1.2.1 Abstract Data Type and data structure

- **What is a data type（数据类型）?**
  - **Class of data objects（某一类数据对象） that have the same properties（属性）**
  - **c 语言中的基本数据类型：int， char， float， double**
  - **构造数据类型：数组，结构体，共用体， 枚举类型等**



Abstract Data Type

- ADT = Data + Relation + Operations

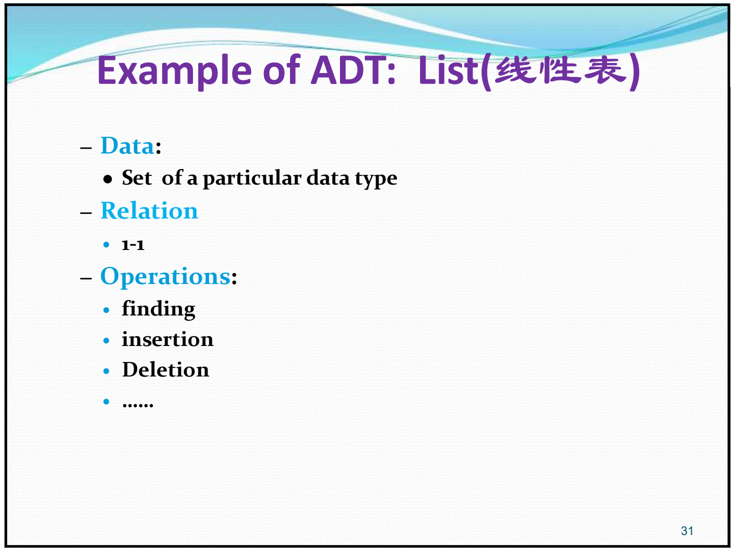
ADT (抽象数据类型)可用

(D，R，O) 三元组表示

其中：D 是数据对象(数据集)

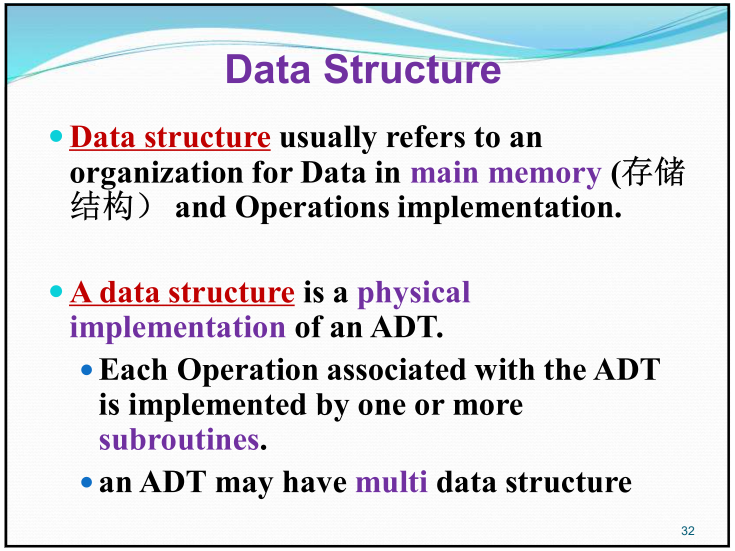
R 是 D上的关系集　(逻辑结构)

O 是对 D 的基本操作集



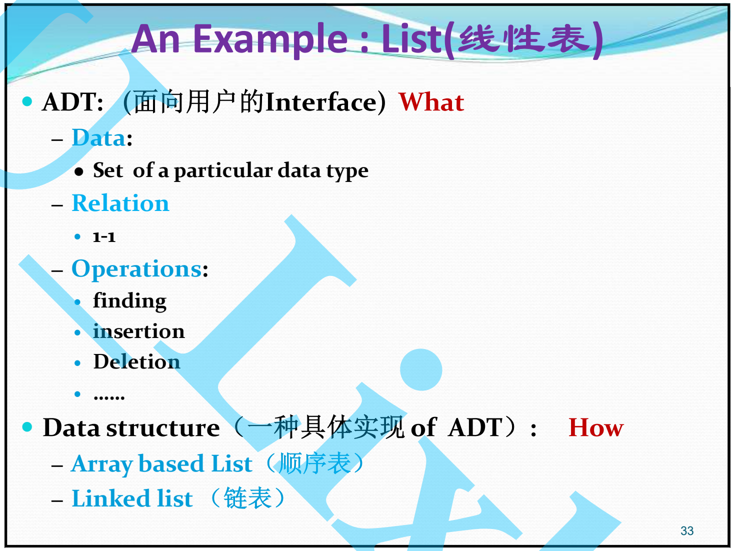
Example of ADT: List(线性表)

- Data:
  - Set of a particular data type
- Relation
  - 1-1
- Operations:
  - finding
  - insertion
  - Deletion
  - ……



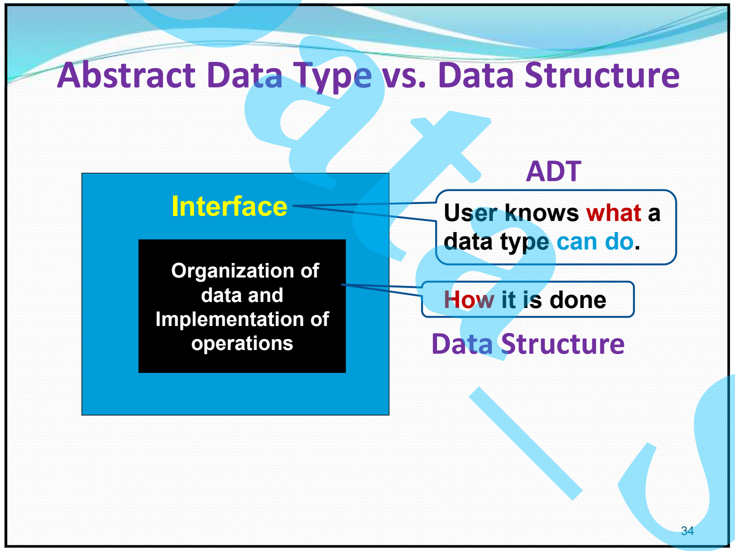
Data Structure

- Data structure usually refers to an organization for Data in main memory (存储结构） and Operations implementation.
- A data structure is a physical implementation of an ADT.
  - Each Operation associated with the ADT is implemented by one or more subroutines.
  - an ADT may have multi data structure

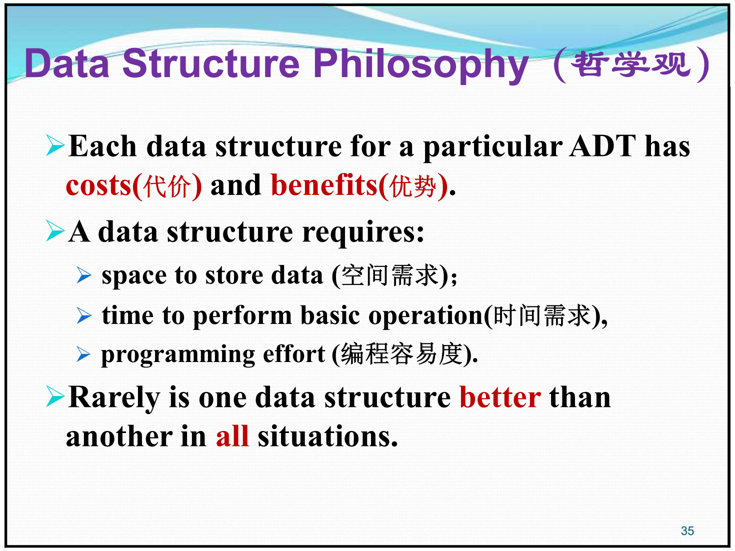


An Example : List(线性表)

- ADT: (面向用户的Interface) What
  - Data:
    - Set of a particular data type
  - Relation
    - 1-1
  - Operations:
    - finding
    - insertion
    - Deletion
    - ……
- Data structure（一种具体实现 of ADT）: How
  - Array based List（顺序表）
  - Linked list（链表）

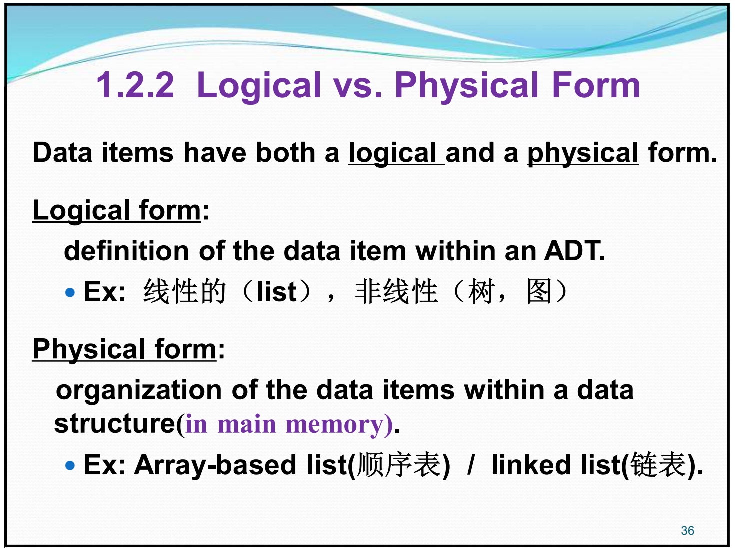


Abstract Data Type vs. Data Structure



Data Structure Philosophy（哲学观）

- Each data structure for a particular ADT has costs(代价) and benefits(优势).
- A data structure requires:
  - space to store data (空间需求);
  - time to perform basic operation(时间需求),
  - programming effort (编程容易度).
- Rarely is one data structure better than another in all situations.



1.2.2 Logical vs. Physical Form

Data items have both a logical and a physical form.

Logical form:

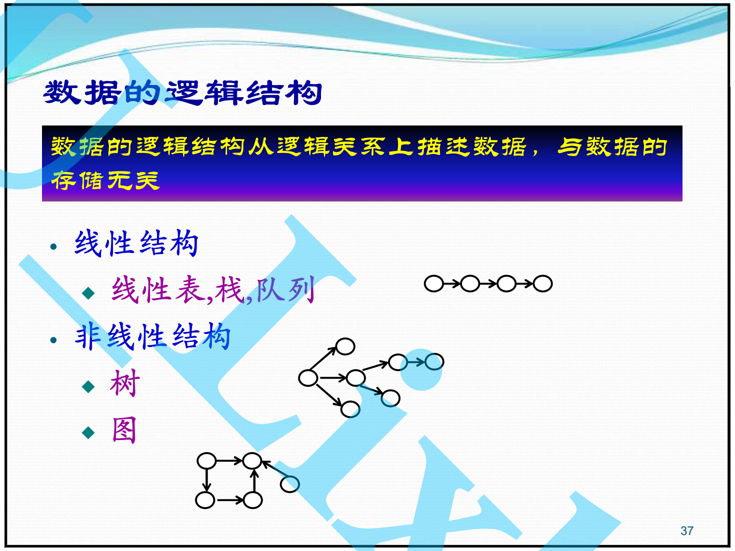
definition of the data item within an ADT.

- Ex: 线性的（list），非线性（树，图）

Physical form:

organization of the data items within a data structure(in main memory).

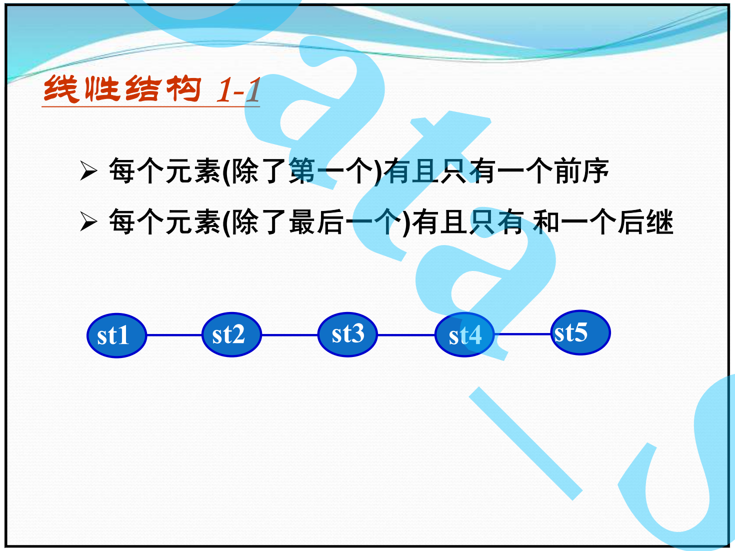
- Ex: Array-based list(顺序表) / linked list(链表).



数据的逻辑结构

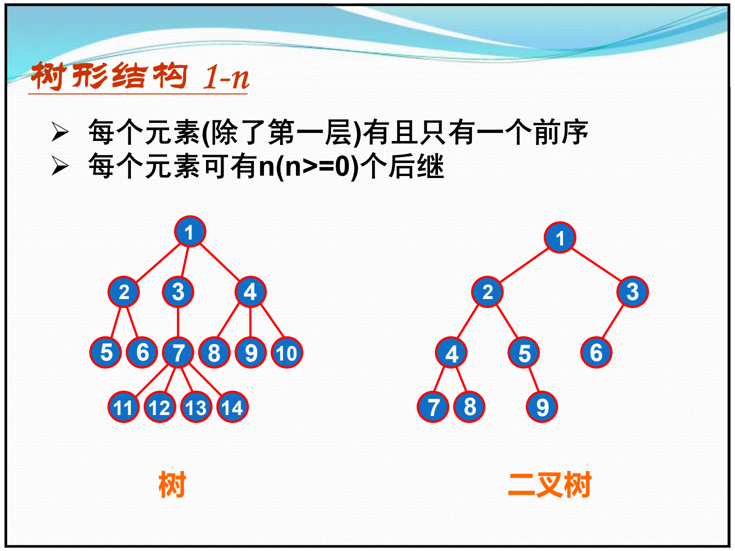
数据的逻辑结构从逻辑关系上描述数据，与数据的存储无关

- 线性结构
  - 线性表,栈,队列
- 非线性结构
  - 树
  - 图



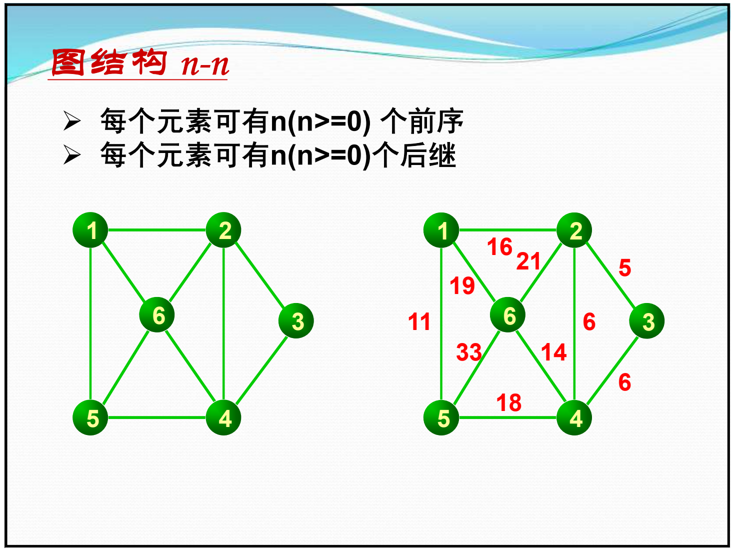
线性结构 *1-1*

- ➢ 每个元素(除了第一个)有且只有一个前序
- ➢ 每个元素(除了最后一个)有且只有 和一个后继



树形结构 *1-n*

- ➢ 每个元素(除了第一层)有且只有一个前序
- ➢ 每个元素可有n(n>=0)个后继

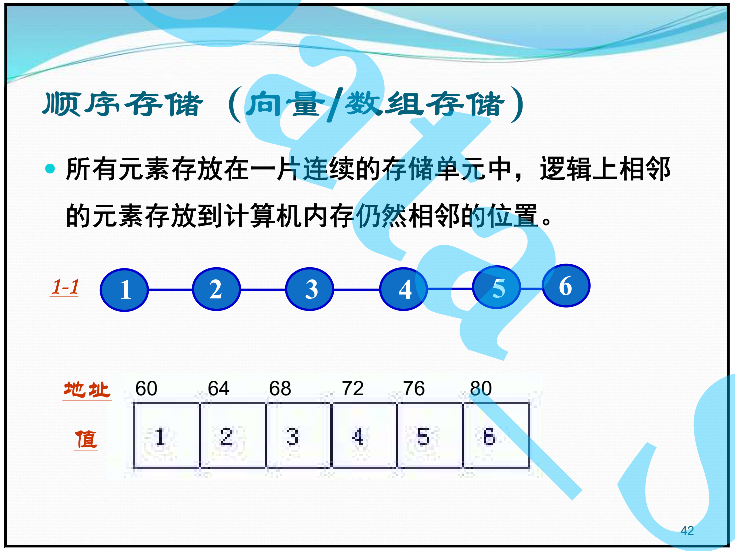


图结构 *n-n*

- ➢ 每个元素可有n(n>=0) 个前序
- ➢ 每个元素可有n(n>=0)个后继

数据的物理/存储结构

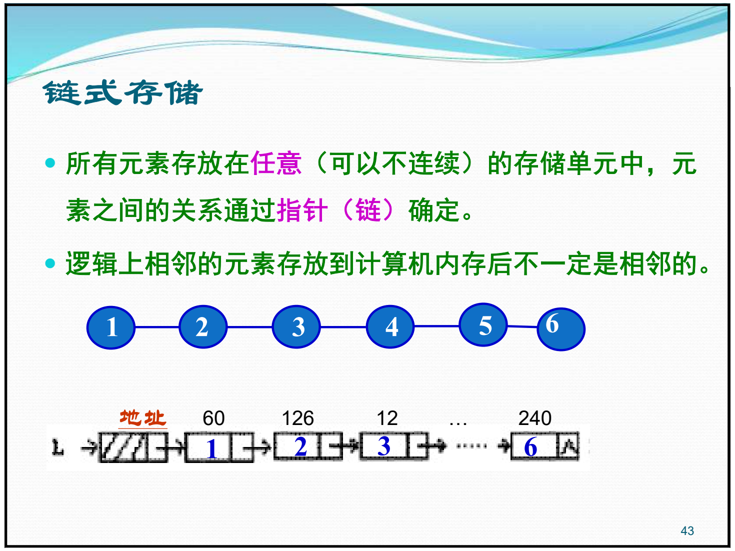
- 数据的物理/存储结构是指数据在计算机内存中的结构
  - ◆ 顺序存储表示
  - ◆ 链接存储表示
  - ◆ 索引存储表示
  - ◆ 散列存储表示



顺序存储（向量/数组存储）

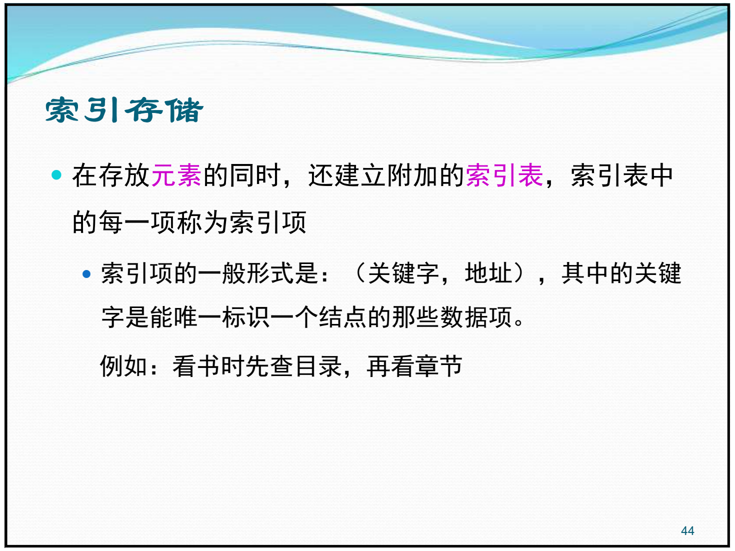
- 所有元素存放在一片连续的存储单元中，逻辑上相邻的元素存放到计算机内存仍然相邻的位置。

地址	60	64	68	72	76	80
值	1	2	3	4	5	6



链式存储

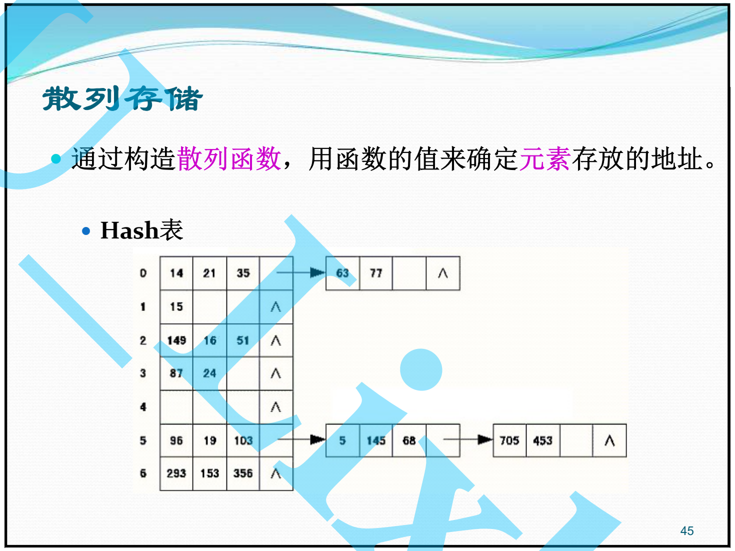
- 所有元素存放在任意（可以不连续）的存储单元中，元素之间的关系通过指针（链）确定。
- 逻辑上相邻的元素存放到计算机内存后不一定是相邻的。



索引存储

- 在存放元素的同时，还建立附加的索引表，索引表中的每一项称为索引项
  - 索引项的一般形式是：（关键字，地址），其中的关键字是能唯一标识一个结点的那些数据项。

    例如：看书时先查目录，再看章节



散列存储

- 通过构造散列函数，用函数的值来确定元素存放的地址。
  - Hash表

0	14	21	35	→	63	77		∧				
1	15			∧								
2	149	16	51	∧								
3	87	24		∧								
4				∧								
5	96	19	103	→	5	145	68	→	705	453		∧
6	293	153	356	∧								

1.2.3 Algorithm(算法) and Program(程序)

- **Algorithm**: a method or a process followed to solve a problem.
  - A recipe (菜谱).
- An algorithm takes the input to a problem and transforms it to the output.
  - A mapping of input to output.
- A problem can have many algorithms.

Algorithm Properties

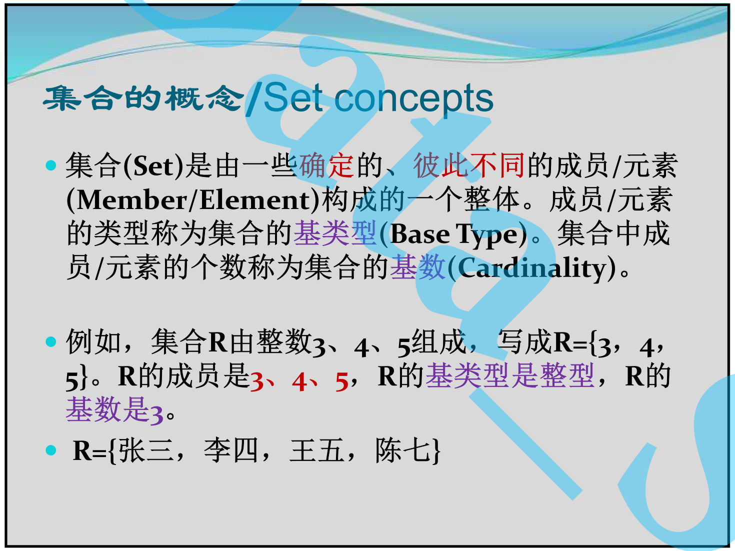
- It must be correct（正确）.
- It must be composed of a series of concrete steps(具体步骤).
- There can be no ambiguity as to which step will be performed next（确定性）.
- It must be composed of a finite number of steps(有穷性)..
- It must terminate（可终止）.
- It must have input and output(有输入输出)

program

- A computer program is an instance(实例), or concrete representation for an algorithm in some programming language.
- Can run

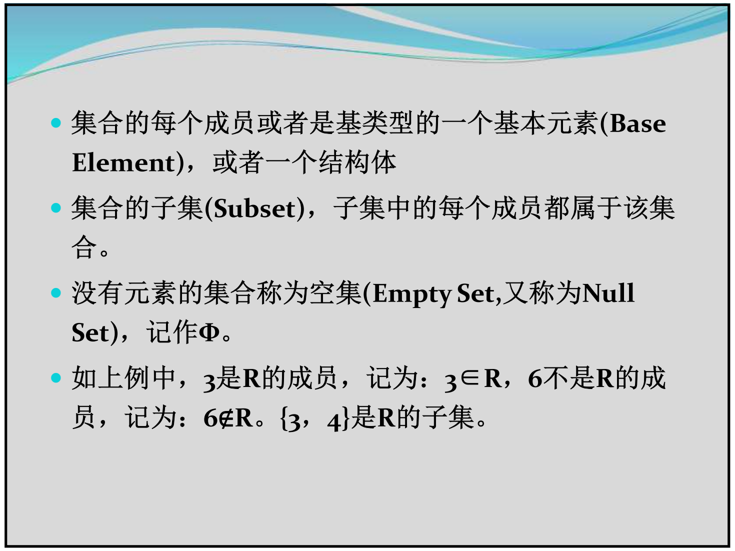
1.3 Mathematical Background

- Set concepts （集合）
- Logarithms（对数）
- Summations （求和）
- Recursion （递归）
- Mathematical Proof Techniques （数学证明法）

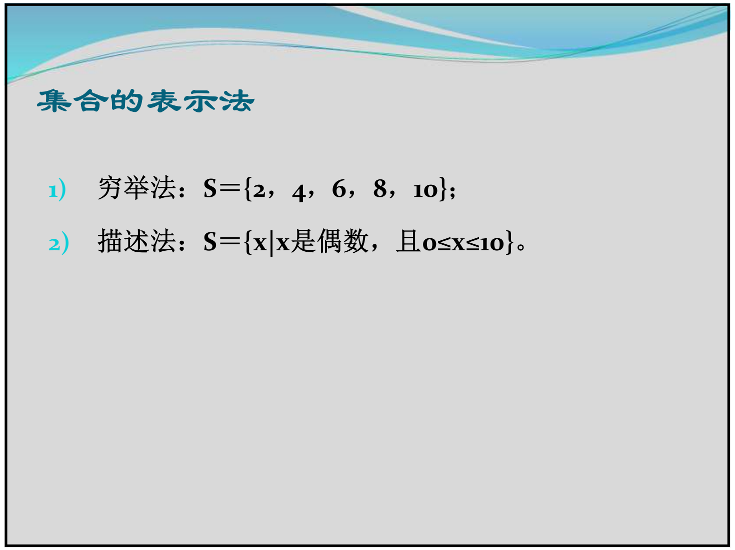


集合的概念/Set concepts

- 集合(Set)是由一些确定的、彼此不同的成员/元素(Member/Element)构成的一个整体。成员/元素的类型称为集合的基类型(Base Type)。集合中成员/元素的个数称为集合的基数(Cardinality)。
- 例如，集合R由整数3、4、5组成，写成R={3，4，5}。R的成员是3、4、5，R的基类型是整型，R的基数是3。
- R={张三，李四，王五，陈七}

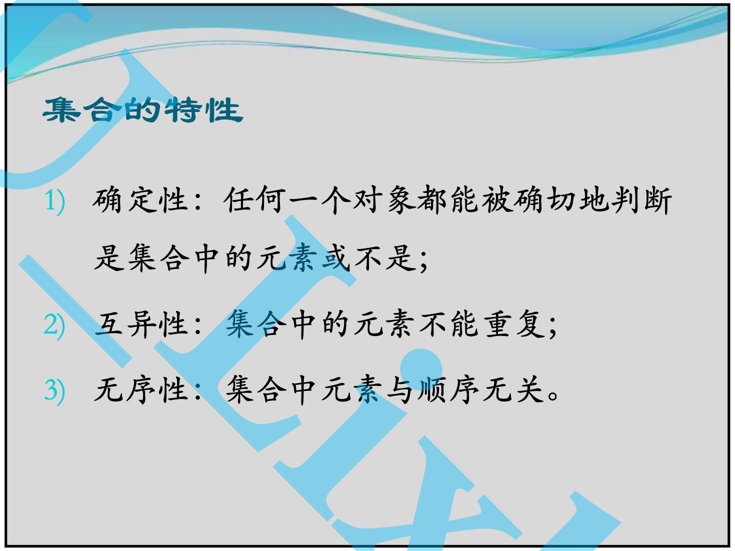


- 集合的每个成员或者是基类型的一个基本元素(Base Element)，或者一个结构体
- 集合的子集(Subset)，子集中的每个成员都属于该集合。
- 没有元素的集合称为空集(Empty Set,又称为Null Set)，记作Φ。
- 如上例中，3是R的成员，记为：3∈R，6不是R的成员，记为：6∉R。{3，4}是R的子集。



集合的表示法

1) 穷举法：S＝{2，4，6，8，10}；
2) 描述法：S＝{x|x是偶数，且0≤x≤10}。



集合的特性

1) 确定性：任何一个对象都能被确切地判断是集合中的元素或不是；
2) 互异性：集合中的元素不能重复；
3) 无序性：集合中元素与顺序无关。

对数/ Logarithms

if $a^b = N$ (a＞0，a≠1)，

then $\log_a N = b$

b：叫做以a为底 N的对数（**Logarithm**)

a： 叫做对数的底数，

N： 叫做真数。

从定义可知，负数和零没有对数。

对数

- 编程人员经常使用对数，它主要有两个用途。
  - 许多程序需要对一些对象进行编码，那么表示N个编码至少需要多少位呢？例如，如果要给1000个学生进行编号（编码），每个学生至少需要10 位（bit）。Why?
  - 对数普遍用于分析把问题分解为更小子问题算法。
    - 对长度为n的有序表的折半查找算法

级数求和/ Summations

- 级数求和 $\sum_{i=1}^{n} f(i)$
- 本书常用： $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

$$\sum_{i=1}^{n} i^2 = \frac{2n^3+3n^2+n}{6}$$

$$\sum_{i=1}^{n} 2^i = 2(2^n - 1)$$

递归/Recursion

- 一个算法调用自己来完成它的部分工作，在解决某些问题时，一个算法需要调用自身。如果一个算法直接调用自己或间接地调用自己，就称这个算法是递归的(Recursive)。
- 汉诺塔问题

递归（续）

- 一个递归算法必须有两个部分：初始部分(Base Case)和递归部分(Recursion Case)。初始部分只处理可以直接解决而不需要再次递归调用的简单输入。递归部分包含对算法的一次或多次递归调用，每一次的调用参数都在某种程度上比原始调用参数更接近初始情况。

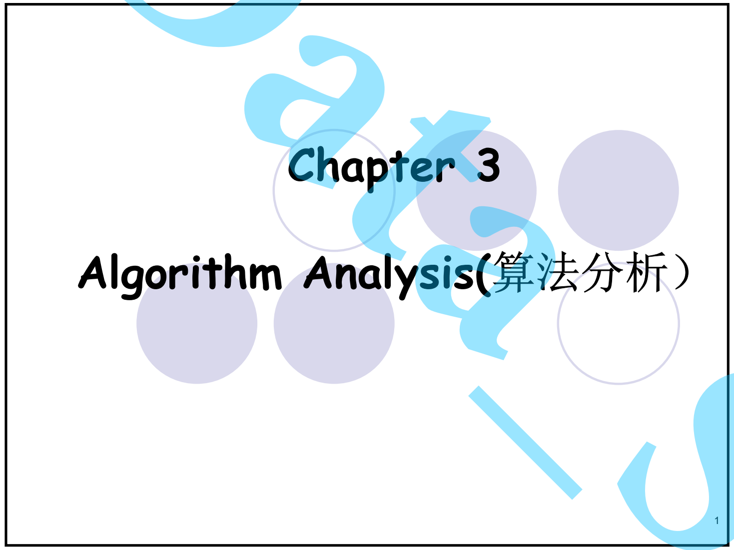
常用数学证明方法

- 数学归纳法：
  - 数学归纳法是一种数学证明方法，典型地用于确定一个表达式在所有自然数范围内是成立的或者用于确定一个其他的形式在一个无穷序列是成立的。
- 反证法：
  - 反证法是属于“间接证明法”一类，是从反面的角度思考问题的证明方法，即：肯定题设而否定结论，从而导出矛盾推理而得。
- 直接证明法

本章我们了解了

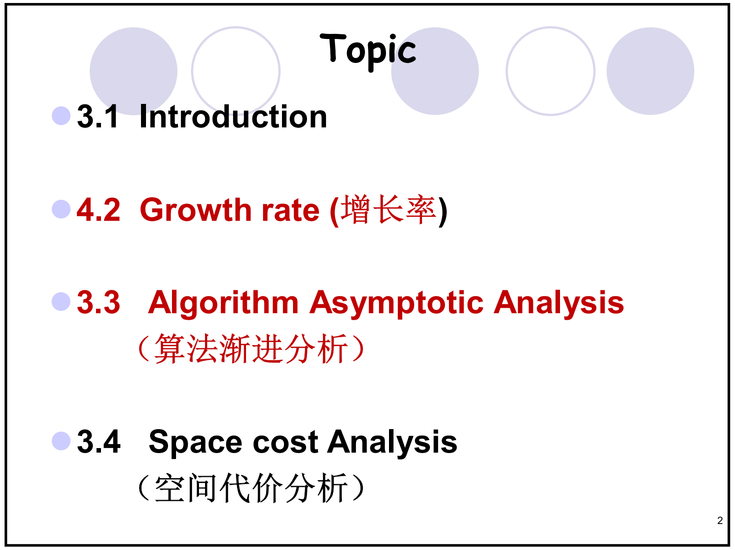
1 ADT && Data Structure
2 Logical vs. Physical Form
3. Algorithm and Program

CHAPTER1 END



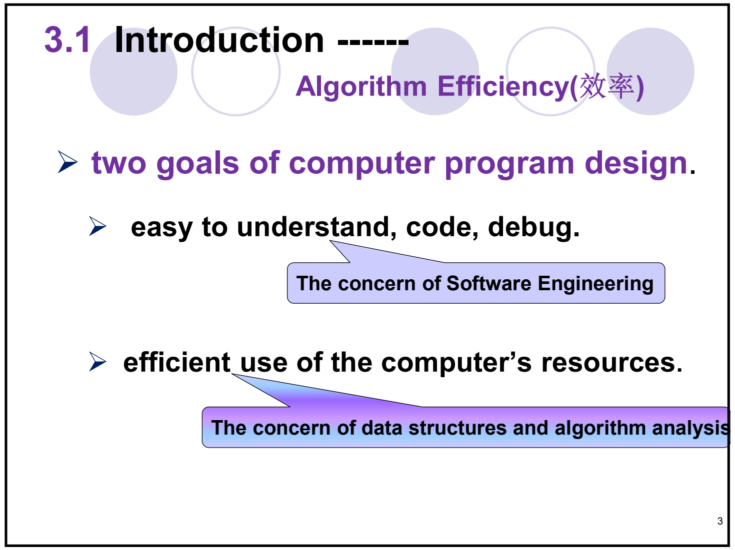
Chapter 3

Algorithm Analysis(算法分析）



Topic

- 3.1 Introduction
- 4.2 Growth rate (增长率)
- 3.3 Algorithm Asymptotic Analysis
  （算法渐进分析）
- 3.4 Space cost Analysis
  （空间代价分析）



3.1 Introduction ------

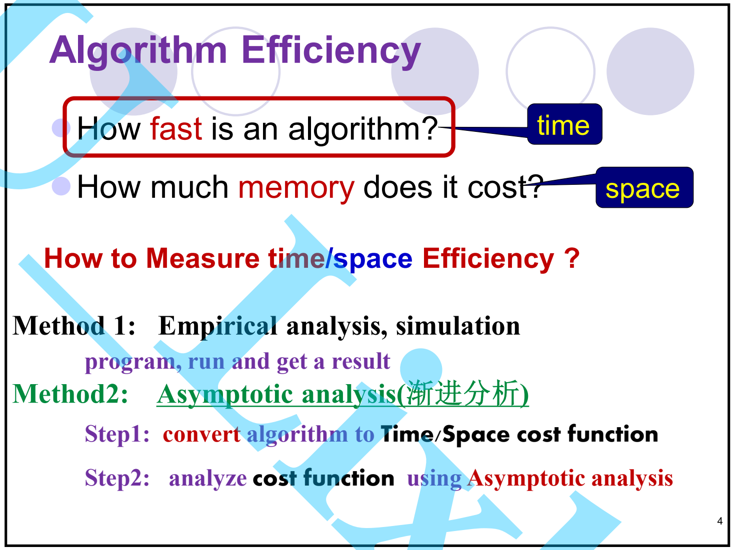
Algorithm Efficiency(效率)

- ➢ two goals of computer program design.
  - ➢ easy to understand, code, debug.

    The concern of Software Engineering

  - ➢ efficient use of the computer’s resources.

    The concern of data structures and algorithm analysis



Algorithm Efficiency

- How fast is an algorithm? — time
- How much memory does it cost? — space

How to Measure time/space Efficiency ?

Method 1: Empirical analysis, simulation

program, run and get a result

Method2: Asymptotic analysis(渐进分析)

Step1: convert algorithm to Time/Space cost function

Step2: analyze cost function using Asymptotic analysis

Algorithm Time cost function(时间代价函数)

- **General format(通用形式):** *f(n)*

n is the size of a problem (the number that determines the size of input data）/问题规模

```
int sum(int array[], int n)
{     int sum=0;
      for (int i=0; i<n; i++)
          sum=sum+array[i];
      return  sum;
}
```

$f(n) = n+2$

```
int search(int K， int array[], int n)
{
      for (int i=0; i<n; i++)
         if (K== array[i])
            return i;
      return -1;
}
```

$f(n) =\ ???$

Best, Average, Worst Cases

- **Best case:** $f_{best}(n)$ given *n*, *f(n)* is smallest.
- **Worst case:** $f_{worst}(n)$ given *n*, *f(n)* is largest.
- **Average case:** $f_{ave}(n)$ given *n*, *f(n)* in between.

请思考

问题规模(n)一定时，什么导致了 $f_{best}(n)$和$f_{worst}(n)$可能不同？

```
// sum an array[]
int sum(int array[], int n)
{     int sum=0;
      for (int i=0; i<n; i++)
          sum=sum+array[i];
      return  sum;
}
```

Best case: $f_{best}(n)$ = n+2
Worst case: $f_{worst}(n)$ = n+2
Average case: $f_{ave}(n)$ = n+2

```
// search K in array[]
int search(int K， int array[ ], int n)
{
      for (int i=0; i<n; i++)
         if (K== array[i])
            return i;
      return -1;
}
```

Best case: K at first position. Cost(Compare times) is 1
Worst case: K at last position or not in. cost is *n*
Average case: cost (*n+1*)/2

一段代码中导致 $f_{best}(n)$， $f_{worst}(n)$不同的是什么语句？

两个常见误解关于 best case and worst case

- The best case for my algorithm is $n=1$， because that is the fastest ✗
- The worst case for my algorithm is $n=\infty$ because that is the slowest ✗

输入数据具体值及其顺序的不确定性导致$f_{best}(n)$，$f_{worst}(n)$不同

一段代码中导致 $f_{best}(n)$，$f_{worst}(n)$不同的主要原因是代码中有分支结构。

Worst case refers to the worst input from among the choices for possible inputs of a given size.
while best case refers to the best input from among the choices for possible inputs of the same given size. √

3.2 Growth rate (增长率)

- 做算法性能分析时关心的不是 $f(n)$的具体形式，而是 $f(n)$的 *growth rate(增长率)*

即： n 增长时，代价函数 $f(n)$的增长速率

阿凡提与巴依的故事

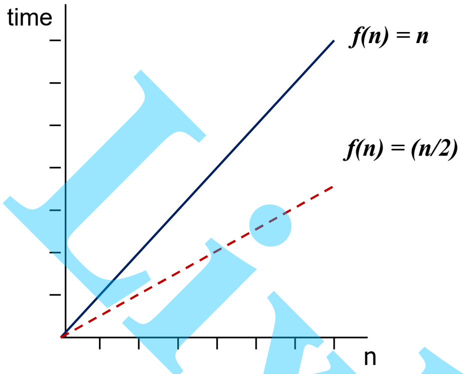
尤其关心 当 n很大很大时的f(n) 的值

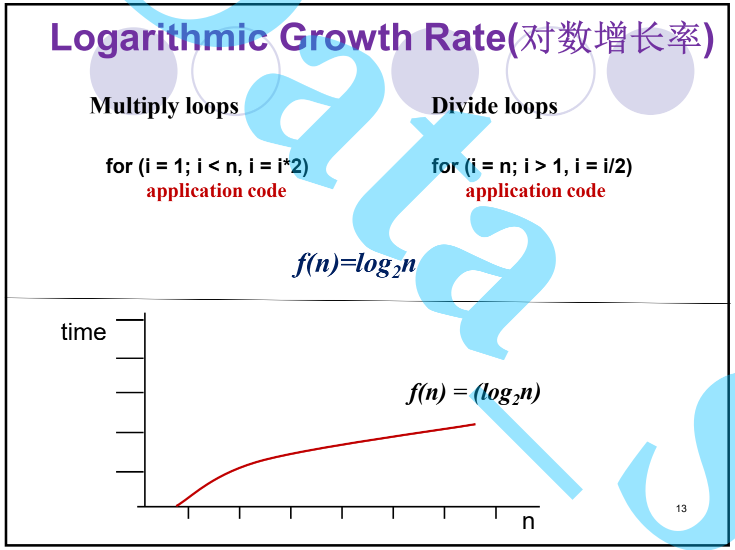
增长率越大，当 n很大很大时的代价函数值越大

Linear Growth Rate(线性增长率)

for (i = 1; i <= n, i++) application code	for (i = 1; i <= n, i=i+2) application code
循环次数 n	循环次数n/2
$f(n) = n$	$f(n) = n/2$

Linear Growth Rate





Logarithmic Growth Rate(对数增长率)

Multiply loops

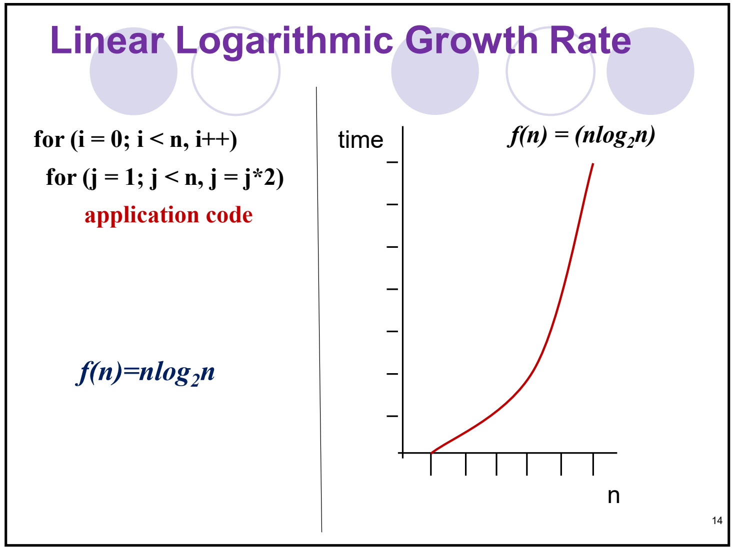
```
for (i = 1; i < n, i = i*2)
    application code
```

Divide loops

```
for (i = n; i > 1, i = i/2)
    application code
```

$f(n)=log_2 n$

13

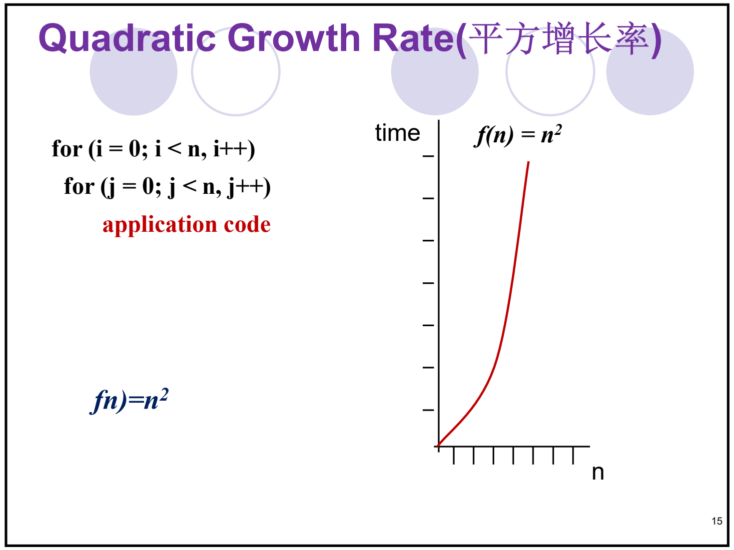


Linear Logarithmic Growth Rate

```
for (i = 0; i < n, i++)
  for (j = 1; j < n, j = j*2)
    application code
```

$f(n)=nlog_2 n$

14

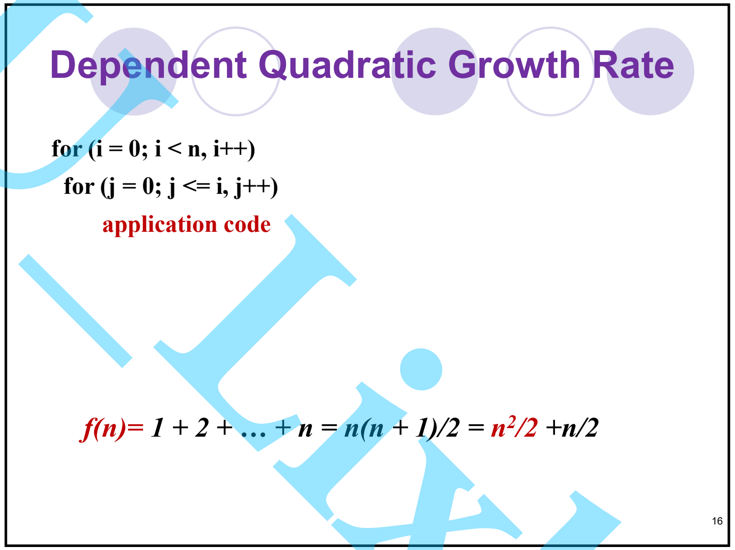


Quadratic Growth Rate(平方增长率)

```
for (i = 0; i < n, i++)
  for (j = 0; j < n, j++)
    application code
```

$fn)=n^2$

15

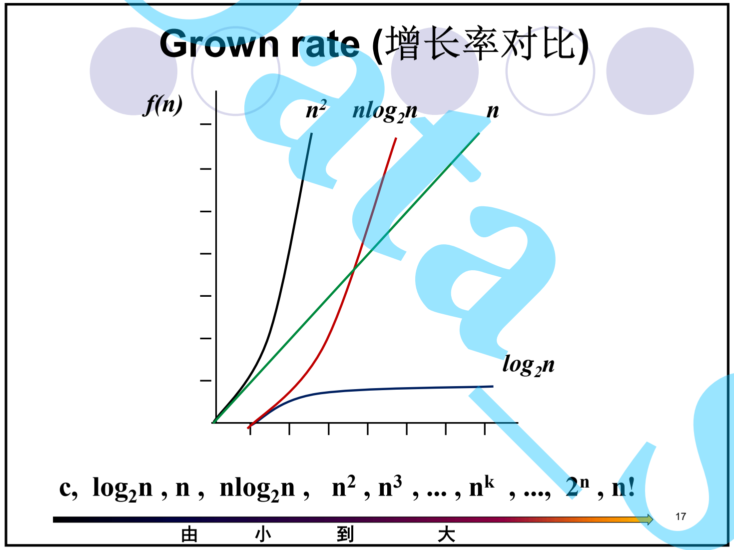


Dependent Quadratic Growth Rate

```
for (i = 0; i < n, i++)
  for (j = 0; j <= i, j++)
    application code
```

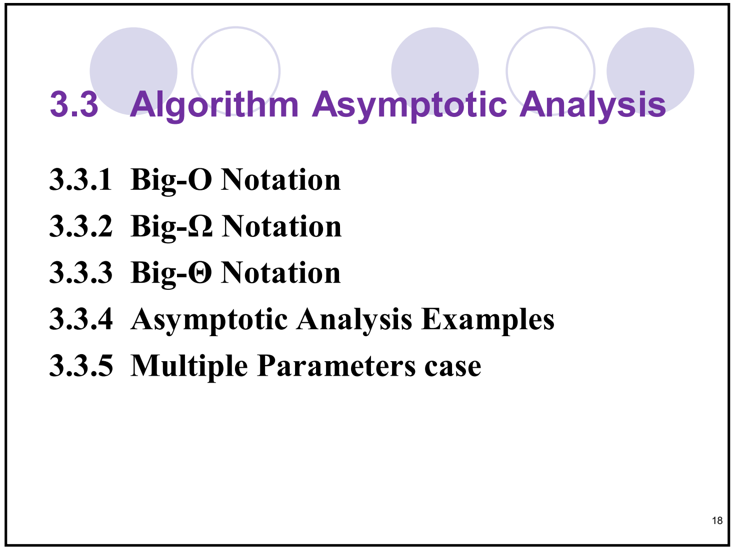
$f(n)= 1 + 2 + … + n = n(n + 1)/2 = n^2/2 +n/2$

16



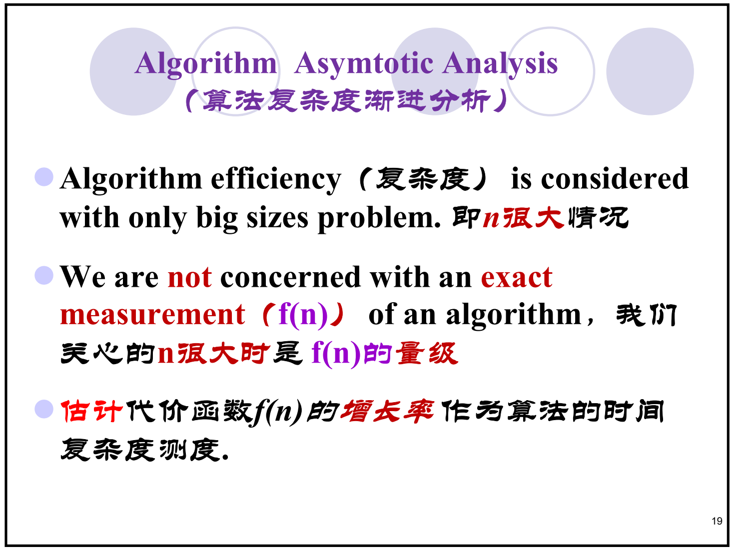
Grown rate (增长率对比)

c, log_2n , n , $nlog_2n$, n^2 , n^3 , ... , n^k , ..., 2^n , n!



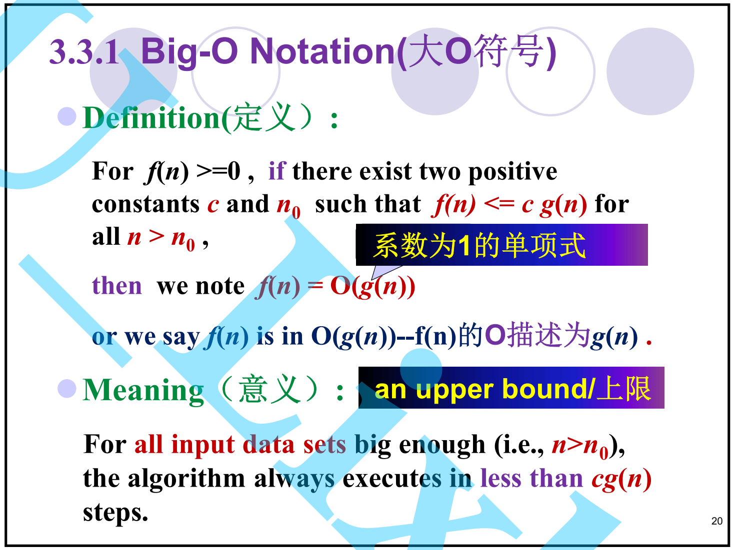
3.3 Algorithm Asymptotic Analysis

3.3.1 Big-O Notation
3.3.2 Big-Ω Notation
3.3.3 Big-Θ Notation
3.3.4 Asymptotic Analysis Examples
3.3.5 Multiple Parameters case



Algorithm Asymtotic Analysis
（算法复杂度渐进分析）

- Algorithm efficiency（复杂度） is considered with only big sizes problem. 即*n*很大情况
- We are not concerned with an exact measurement（f(n)） of an algorithm，我们关心的n很大时是 f(n)的量级
- 估计代价函数$f(n)$的增长率作为算法的时间复杂度测度.



3.3.1 Big-O Notation(大O符号)

- **Definition(定义）：**

For $f(n) >=0$, if there exist two positive constants c and n_0 such that $f(n) <= c\, g(n)$ for all $n > n_0$,

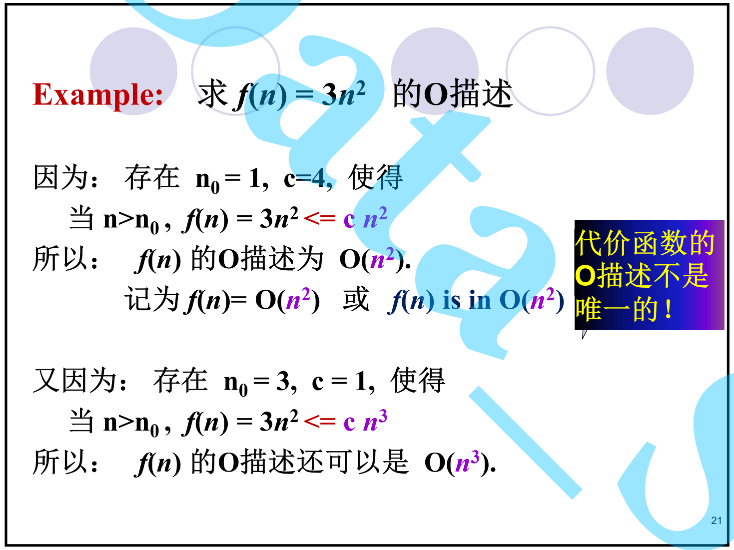
系数为1的单项式

then we note $f(n) = O(g(n))$

or we say $f(n)$ is in $O(g(n))$--f(n)的O描述为$g(n)$.

- **Meaning（意义）：** an upper bound/上限

For all input data sets big enough (i.e., $n>n_0$), the algorithm always executes in less than $cg(n)$ steps.



Example: 求 $f(n) = 3n^2$ 的O描述

因为： 存在 $n_0 = 1$, $c=4$, 使得

当 $n>n_0$, $f(n) = 3n^2 <= c\ n^2$

所以： $f(n)$ 的O描述为 $O(n^2)$.

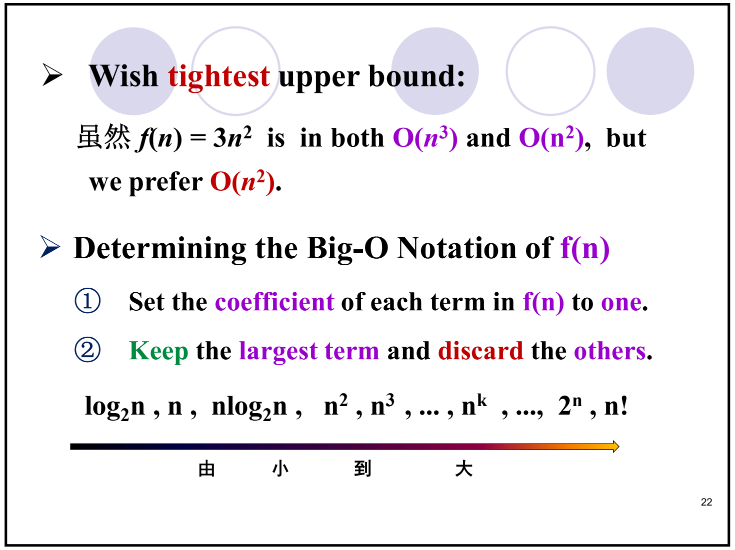
记为 $f(n)= O(n^2)$ 或 $f(n)$ is in $O(n^2)$

代价函数的O描述不是唯一的！

又因为： 存在 $n_0 = 3$, $c = 1$, 使得

当 $n>n_0$, $f(n) = 3n^2 <= c\ n^3$

所以： $f(n)$ 的O描述还可以是 $O(n^3)$.



- **Wish tightest upper bound:**

虽然 $f(n) = 3n^2$ is in both $O(n^3)$ and $O(n^2)$, but we prefer $O(n^2)$.

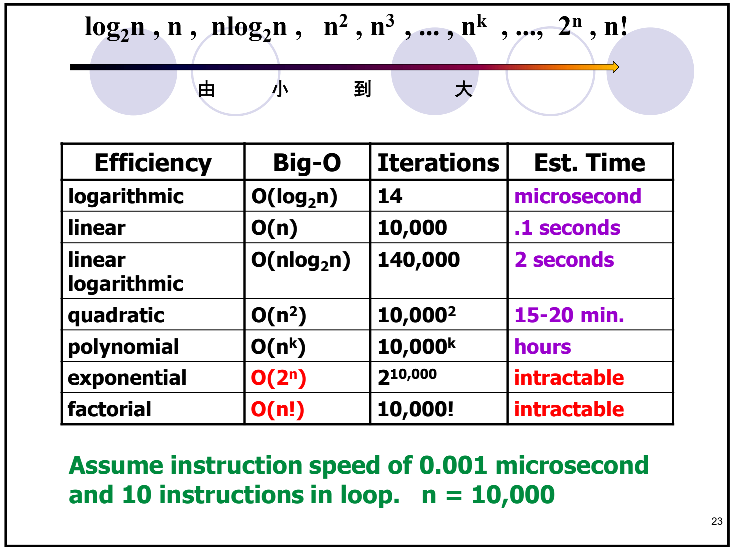
- **Determining the Big-O Notation of f(n)**

① Set the coefficient of each term in f(n) to one.

② Keep the largest term and discard the others.

$\log_2 n$, n , $n\log_2 n$, n^2 , n^3 , ... , n^k , ..., 2^n , $n!$

由 小 到 大

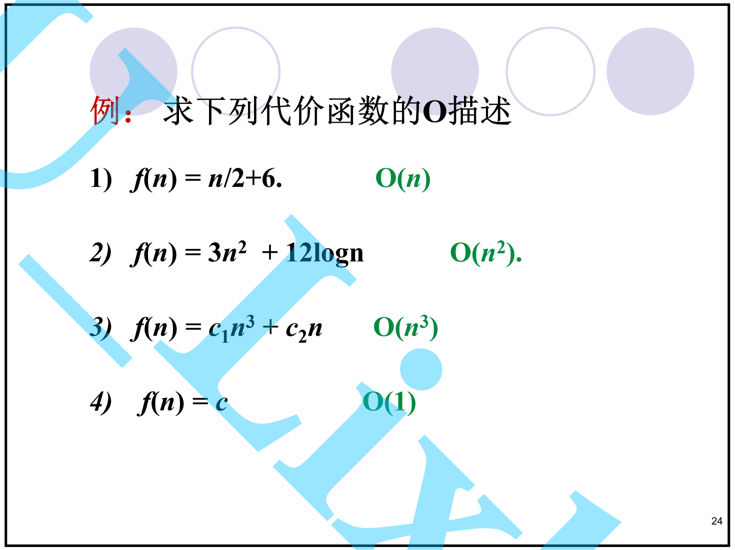


$\log_2 n$, n , $n\log_2 n$, n^2 , n^3 , ... , n^k , ..., 2^n , $n!$

由 小 到 大

Efficiency	Big-O	Iterations	Est. Time
logarithmic	$O(\log_2 n)$	14	microsecond
linear	O(n)	10,000	.1 seconds
linear logarithmic	$O(n\log_2 n)$	140,000	2 seconds
quadratic	$O(n^2)$	$10{,}000^2$	15-20 min.
polynomial	$O(n^k)$	$10{,}000^k$	hours
exponential	$O(2^n)$	$2^{10{,}000}$	intractable
factorial	O(n!)	10,000!	intractable

Assume instruction speed of 0.001 microsecond and 10 instructions in loop. n = 10,000



例： 求下列代价函数的O描述

1) $f(n) = n/2+6$. $O(n)$

2) $f(n) = 3n^2 + 12\log n$ $O(n^2)$.

3) $f(n) = c_1n^3 + c_2n$ $O(n^3)$

4) $f(n) = c$ $O(1)$

3.3.2 Big-Ω Notation(大Ω符号)—an Lower bound

- **Definition(定义):**

For $f(n) >= 0$, if there exist two positive constants c and n_0 such that $f(n) >= c\ g(n)$ for all $n > n_0$,

then we note $f(n) = \Omega\ (g(n))$ 系数为1的单项式

- **Meaning(意义):** an lower bound/下限

For all data sets big enough (i.e., $n > n_0$), the algorithm always executes in more than $cg(n)$ steps.

Example: 求 $f(n) = 3n^2$ 的Ω描述

因为：存在 $n_0 = 1$, $c=2$, 使得
当 $n>n_0$, $f(n) = 3n^2 >= c\ n^2$
所以： $f(n)$ 的Ω描述为 $\Omega(n^2)$. 记为$f(n)= \Omega(n^2)$

又因为：存在 $n_0 = 1$, $c = 1$, 使得
当 $n>n_0$, $f(n) = 3n^2 >= c\ n$
所以： $f(n)$ 的Ω描述还可以是 $\Omega(n)$.

代价函数的Ω描述不是唯一的！

- **Wish tightest(greatest) lower bound**

While $f(n) = 3n^2$ is in both $\Omega\ (n)$ and $\Omega\ (n^2)$,

but we prefer $\Omega(n^2)$.

- **Determining the Big-Ω Notation of f(n)---**

Same method as the Big-O

① Set the coefficient of each term in f(n) to one.

② Keep the largest term and discard the others.

由 小 到 大

$\log_2 n$, n , $n\log_2 n$, n^2 , n^3 , ... , n^k , ..., 2^n , n !

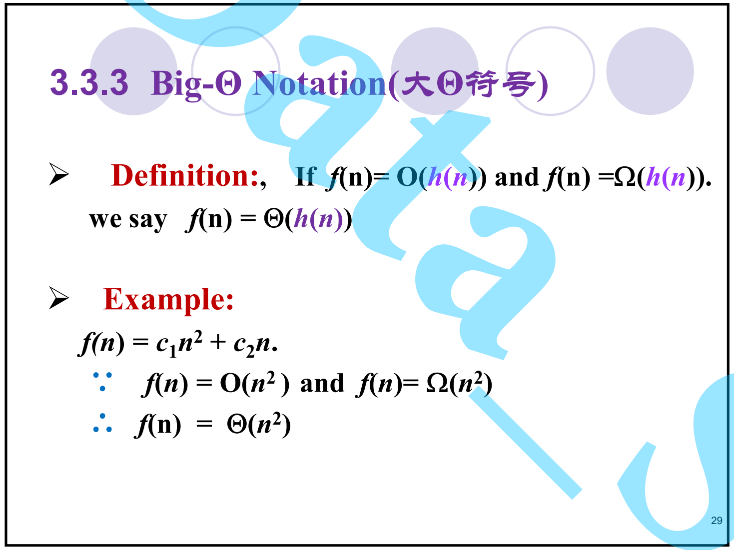
例：求下列代价函数的Ω描述

1） $f(n) = 4n\log n + n/2.$ $\Omega(n\log n)$

2） $f(n) = 7+3n^2$ $\Omega(n^2)$.

3） $f(n) = c_1 n!+c_2 n^2 + c_3$ $\Omega\ (n!)$

4） $f(n) = c$ $\Omega\ (1)$



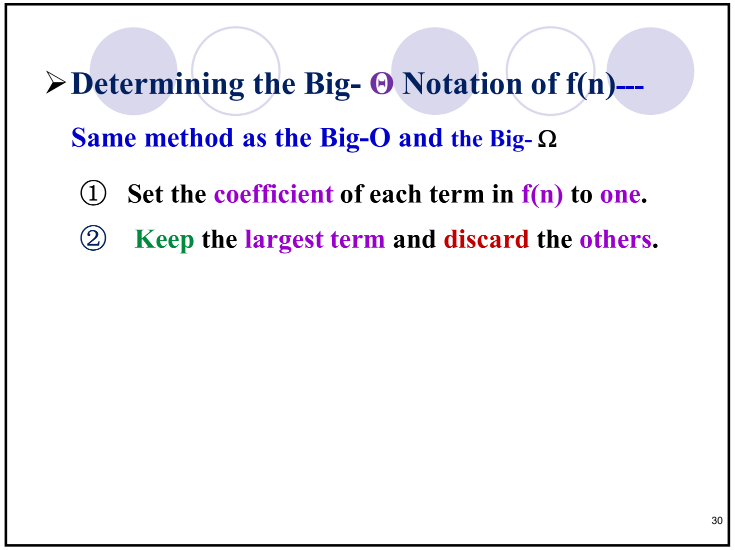
3.3.3 Big-Θ Notation(大Θ符号)

- **Definition:**, If $f(n)= O(h(n))$ and $f(n) =\Omega(h(n))$. we say $f(n) = \Theta(h(n))$
- **Example:**

$f(n) = c_1n^2 + c_2n$.

∵ $f(n) = O(n^2)$ and $f(n)= \Omega(n^2)$

∴ $f(n) = \Theta(n^2)$

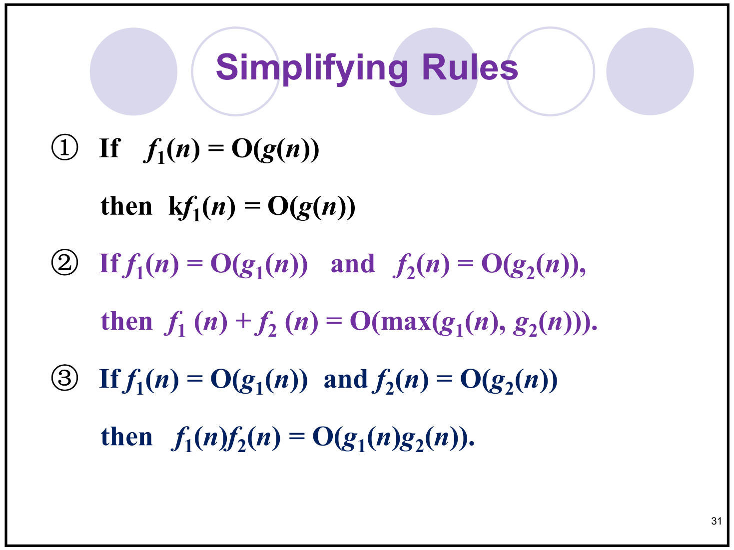


- **Determining the Big- Θ Notation of f(n)---**

Same method as the Big-O and the Big- Ω

① Set the coefficient of each term in f(n) to one.

② Keep the largest term and discard the others.



Simplifying Rules

① If $f_1(n) = O(g(n))$

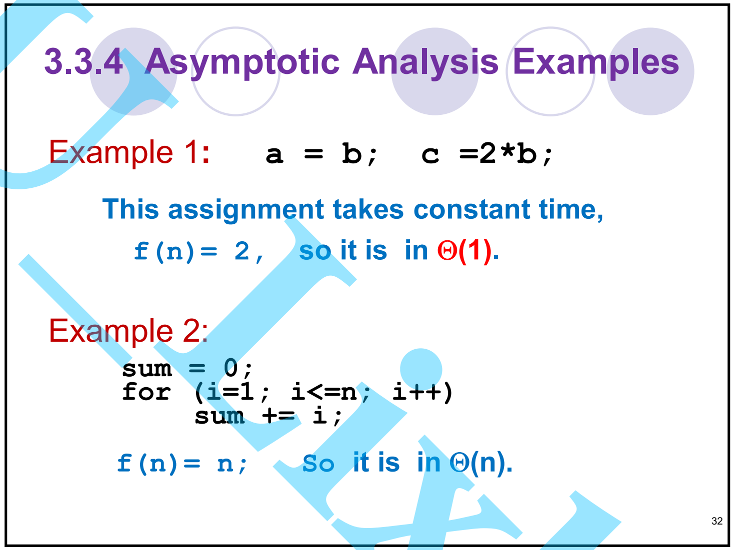
then $kf_1(n) = O(g(n))$

② If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$,

then $f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$.

③ If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$

then $f_1(n)f_2(n) = O(g_1(n)g_2(n))$.



3.3.4 Asymptotic Analysis Examples

Example 1: `a = b; c =2*b;`

This assignment takes constant time, `f(n)= 2,` so it is in Θ(1).

Example 2:

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

`f(n)= n;` So it is in Θ(n).

Example 3:

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

for (k=0; k<n; k++)
        A[k] = k;
```

∵ $f(n) = n(n+1)/2 + n = n^2/2+3n/2$

∴ it is in $\Theta(n^2)$

Example 4:

```
sum1 = 0;
for (i=1; i<=n; i++)
   for (j=1; j<=n; j++)
      sum1++;
sum2 = 0;
for (i=1; i<=n; i++)
   for (j=1; j<=i; j++)
      sum2++;
```

∵ $f(n) = n^2 + n(n+1)/2 = 3n^2/2+n/2$

∴ it is in $\Theta(n^2)$

Example 5:

```
sum1 = 0;
for (k=1; k<n; k*=2)
  for (j=0; j<n; j++)
    sum1++;
sum2 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=k; j++)
    sum2++;
```

$f_1(n) = n\log n$

$\Theta(n \log n)$.

$f_2(n)=1+2+2^2+2^3+\ldots+2^{\log n} = 2n-1$

$\Theta(n)$

$\Theta(n \log n)$.

Example 6 (从有序的数组array中查找K)

顺序查找

```
int  Search1(int K,  int A[],  int n) {
  for (int i=0; i<n; i++)
    if (K==A[i] ) return  i ;
    else if(K>A[i]) return  -1 ;
}
```

Which one is better? Why?

```
int  Search2(int K, int A[], int n) {
   int  l = -1;  int r = n;     // l, r are beyond array bounds
   while ( (l+1) != r)  // Stop when l, r meet
   {
      int i = (l+r)/2;     // Check middle
      if (K < array[i])    r = i;   // Left half
      else if (K == array[i]) return i; // Found it
      else if (K > array[i])    l = i;   // Right half
   }
   return n; // Search value not in array
}
```

折半查找

Example 7

Consider the following C++ code fragment.

```
    x=191; y=200;
    while(y > 0)
        If (x > 200)
          { x=x-10;   y--;}
        else x++;
```

What is its asymptotic time complexity? (   )

A. O(1)

B. O(n)

C. O(n^2)

D. O(n^3)

一些常见误解关于best/worst case 和 O/Ω符号

- Confusing worst case with upper bound(O) ×
- Confusing best case with lower bound(Ω) ×

✓ O/Ω符号都是针对算法时间代价函数f(n)进行的渐进分析，

✓ 而f(n)可以是f_{best}(n), 可以是f_{worst}(n).

✓ f_{best}(n)有它的upper bound(O) 和lower bound(Ω)

✓ f_{worst}(n)也有它的upper bound(O) 和lower bound(Ω) √

✓ 而实际上，我们常常只是对f_{ave}(n)做O/Ω渐进分析

3.4 Space cost Analysis

- Space cost can also be analyzed with asymptotic complexity analysis.
  - Time: Algorithm
  - Space: Data Structure S(n)

这一章我们学到了

- Algorithm Efficiency
  - Time, space
- Cost function of Algorithm f(n)
  - Best case: $f_{best}(n)$, worst case: $f_{worst}(n)$, average case: $f_{ave}(n)$
- Grown rate
  - c, $\log_2 n$, n, $n\log_2 n$, n^2, n^3, ..., n^k, ..., 2^n, n!
- O-Notation
  - upper bound
- Ω-Notation
  - lower bound
- Θ-Notation

本章作业

- 3.3
- 3.9
- 3.12 (b) (d) (g) (h)

Chapter3 end