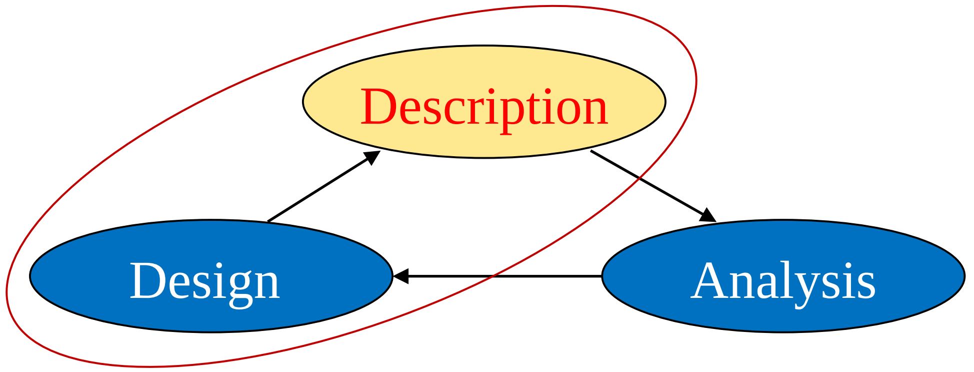
Chapter 2

Framework for analyzing algorithms

songyou@buaa.edu.cn

2 Framework for analyzing algorithms



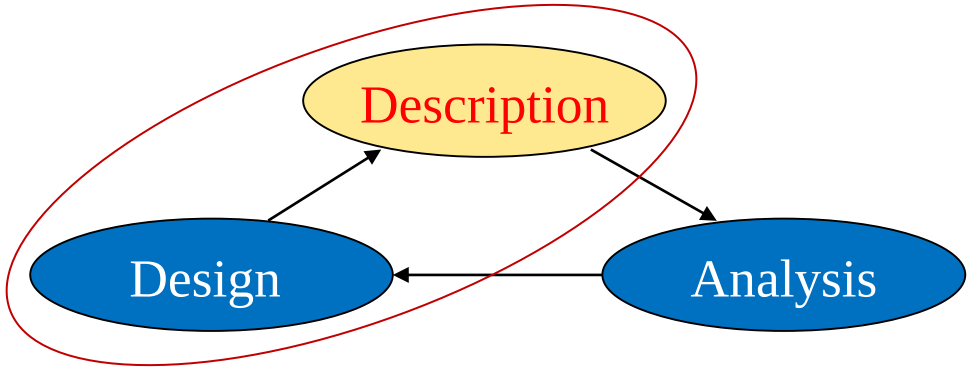
- Description： See how to describe algorithms in pseudocode, C/C++ and argue the correctness of that. （如何描述算法）
- Analysis： Begin using asymptotic notation to express running-time analysis. （使用一致性符号来分析算法的运算时间）
- Design： for example, the technique of “divide and conquer” in the context of merge sort. （例如，通过分而治之方法进行 merge sort）
- Start using framework for describing and analyzing algorithms （描述和分析算法的框架）
- Examine two algorithms for sorting: insertion sort and merge sort

2 Framework for analyzing algorithms

Design • Description

• Correctness

Analysis • Effeciency (running time)



- How
- Why
- What

- Who are you
- Where are you from
- Where will you go

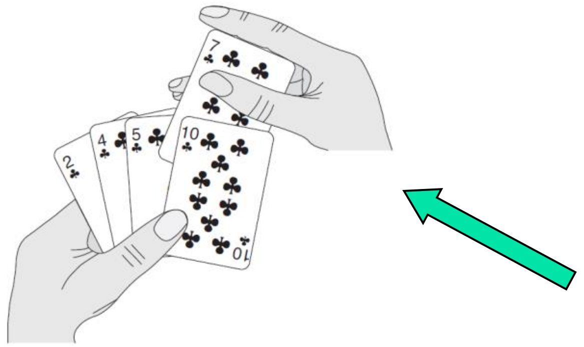
2 Framework for analyzing algorithms

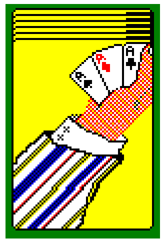
2.1 Insertion sort

Problem: to sort a sequence of numbers into nondecreasing order

Input: A sequence of n numbers $<a_1, a_2, \ldots, a_n>$

Output: A permutation (reordering) $<a_1', a_2', \ldots, a_n'>$ of the input sequence such that $a_1' \leq a_2' \leq \ldots \leq a_n'$





2.1 Insertion sort -- **Description**

- Pseudocode

  That is similar in many respects to C, Pascal, or Java, py, …

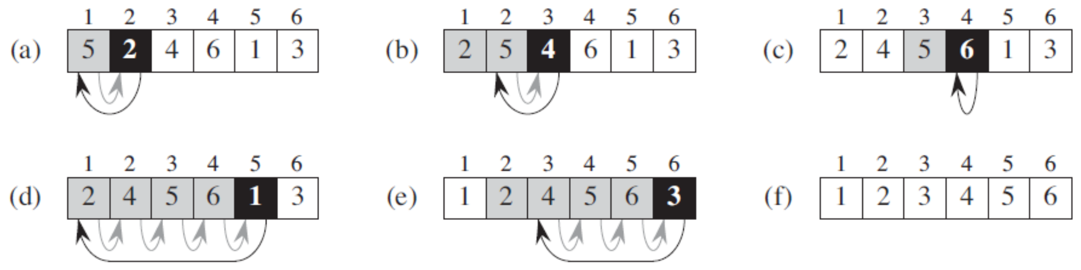
- Differences between psuedocode and real code
  - Psuedocode is clear and concise to specify a given algorithm（清晰、准确）
  - Pseudocode is not typically concerned with issues of software engineering (data abstraction, modularity(模块化), error handling)
    （不用考虑软件工程相关的技术细节，如数据抽象、模块化、容错处理）

用伪代码可以体现算法的本质
永远不会过时（因为算法永远不会过时）

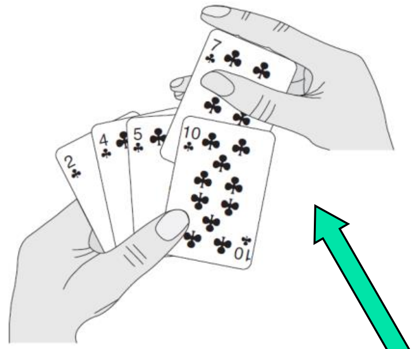
2.1.1 Pseudocode conventions

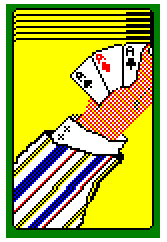
- Indentation（缩排）indicates block structure.
- The looping constructs (**while**, **for**, **repeat**) and the conditional constructs (**if**, **then**, **else**) have interpretations similar to those in Pascal, C.
- "// " or " ▷ " indicates a comment.
- A multiple assignment $i \leftarrow j \leftarrow e$ is equivalent to $j \leftarrow e$ *then* $i \leftarrow j$.
- Variables are local to the given procedure.
- Array elements access: $A[i]$; $A[1 .. j] = < A[1], A[2], \ldots, A[i] >$
- Compound data are typically organized into objects.
- Parameters are passed to a procedure by value.
- The boolean operators "and" and "or" are short circuiting: for example "x and (or) y": whether or not evaluate y rely on the evaluating of x.

2.1 Insertion sort -- **Description**



(a) 1 2 3 4 5 6 | 5 2 4 6 1 3
(b) 1 2 3 4 5 6 | 2 5 4 6 1 3
(c) 1 2 3 4 5 6 | 2 4 5 6 1 3
(d) 1 2 3 4 5 6 | 2 4 5 6 1 3
(e) 1 2 3 4 5 6 | 1 2 4 5 6 3
(f) 1 2 3 4 5 6 | 1 2 3 4 5 6





```
INSERTION-SORT(A)
1       for( j = 2; j <=length[A]; j++)
2       {       key = A[j]
3               // Insert A[j] into the sorted sequence A[1 .. j-1]
4               i = j-1
5               while( i > 0 && A[i] > key)
6               {       A[i+1] = A[i]  // A[i] 比 key 大，  A[i] 后移
7                       i = i-1          // 分析元素 A[i-1]
8               }
9               A[i+1] = key          // 把 key 放到正确的位置
10      }
```

Is the algorithm correct ?

2.1.2 Loop invariants and the **correctness** of insertion sort

- **Loop invariants（循环不变量）**

  Loop invariant: At the start of each iteration of **for** loop, the subarray $A[1 .. j\text{-}1]$ consists of the elements originally in $A[1 .. j\text{-}1]$ but in sorted order.
  插入排序算法的循环不变量定义： for循环的每一次循环前，子数组$A[1 .. j\text{-}1]$包括数组原始的前 j 个元素，且$A[1 .. j\text{-}1]$已经排好序。

- **Three properties about a loop invariant**

  （初值、维持、终值）（初始化、保存、终止）

  - **Initialization**: it is true prior to the first iteration of the loop.(base case)
  - **Maintenance**: if it is true before an iteration of the loop, it remains true before the next iteration. (inductive step) ($[i\text{-}1]$ true => $[i]$ true)
  - **Termination**: When the loop terminates, the invariant show that the algorithms is correct. (stopping the "induction" when the loop terminates; the inductive step of mathematical induction is used infinitely. 类似于数学归纳法，但数学归纳法能用于无限情况)

  When the first two properties hold, the loop invariant is true prior to every iteration of the loop. It is similar to mathematical induction.

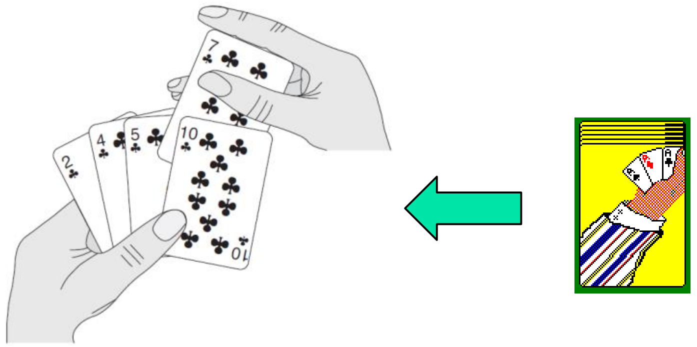
2.1.2 Loop invariants and the **correctness** of insertion sort

Three properties hold for insertion sort

Loop invariant: original $A[1..j\text{-}1]$ is permuted to $A'[1..j\text{-}1]$ but in sorted order.

- **Initialization**: j=2, $A[1 .. j\text{-}1]$ consists of just the single $A[1]$, which is the original element in $A[1]$, and is sorted. Obviously, the loop invariant holds prior to the first iteration of the loop.
- **Maintenance**: $A'[1 .. j\text{-}1]$ is in sorted order, sorting $A[j]$ into $A'[1 .. j\text{-}1]$, s.t. $<a'_1, a'_2, \ldots, a'_k, a_j, a'_{k+1} \ldots, a'_{j-1}>$, obviously, it is sorted.The second property holds for the outer loop.
- **Termination**: when $j = n$+1, $A'[1 .. j\text{-}1] = A'[1 .. n]$ consists of the elements originally in $A[1 .. n]$, but in sorted sorder! Hence, the algorithm is correct.

```
INSERTION-SORT(A)
      for( j = 2; j <=length[A]; j++)
      {    key = A[j]
           i = j-1
           while( i > 0 && A[i] > key)
           {      A[i+1] = A[i]
                  i = i-1
           }
           A[i+1] = key
      }
```



2.2 Analyzing algorithms -- **Effeciency**

Analyzing an algorithm means predicting the resources
（算法分析就是评估或预测资源）

- Resources:
    - computational time (in CPU, I/O, …)
    - space (memory)
    - communication bandwidth
    - primary concern computer hardware
- By analyzing several candidate algorithms for a problem, a most efficient one can be identified, but several inferior are discarded
（算法分析的目的：找出求解问题的最优方案）

2.2 Analyzing algorithms -- Model

几种等价的计算模型：原始递归函数，λ 演算，图灵机，RAM

Computing model (计算模型)

- RAM, Random-access machine 随机存取器（Von-neumann, 冯.诺依曼模型）：带着可随机访问的内存，主要是能用电子元器件做出来。实际普遍使用的计算模型，通过程序数据存储的思想解决了图灵奖的不足。

- 图灵机（七元组的符合体系，抽象的计算机器）：状态自动转移，就是机器指令的例行程序。但指令不能存储起来重复使用，没能形成实现程序的结构设计，由于这两点缺欠，使图灵机还不能成为能够处理各种任务的计算机。
- λ演算（丘奇，数理逻辑意义上）：可以被称为最小的通用程序设计语言。它包括一条变换规则 (变量替换) 和一条函数定义方式，λ演算之通用在于，任何一个可计算函数都能用这种形式来表达和求值。
- 原始递归函数（哥德尔，数学意义上）：一种能停机的递归函数，可以用图灵机计算。

2.2 Analyzing algorithms -- Model

RAM model: Algorithms will be implemented as computer programs.
（算法将作为计算机程序实现）

- RAM, a generic one-processor, instructions are executed one after another, with no concurrent operations.（单处理器，不考虑并行操作）
- RAM contains instructions commonly (basic instructions)
（通常只考虑一些常用的基本操作，如算术运算，位运算，逻辑运算，控制指令等）
  - Arithmetic (add, subtract, multiply, divide, remainder, floor, ceiling)
  - Data movement (load, store, copy)
  - Control (conditional and unconditional branch, subroutine call and return)
- Each such instruction takes a constant amount of time.
（每个基本操作执行的计算时间为常数）

2.2 Analyzing algorithms -- Challenge

Analyzing a simple algorithm can be a challenge

- Mathematics required: probability theory, combinatorics, algebraic dexterity
  （数学功底好，比如概率、组合、代数等知识）
- An ability to identify the most significant terms in a formula
  （洞察能力强，能抓住最重要的部分）
- Summarizing an algorithm in simple formulas
  （抽象思维强，能将算法的本质特征归纳成简单形式）

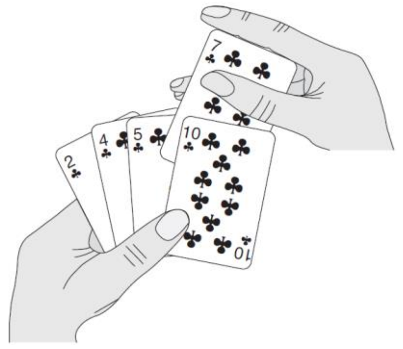
2.2.1 Analysis of insertion sort -- Running time

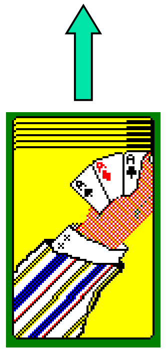
Running time of a program is described as a function of the size of its input.
（通常，程序的运算时间是输入规模的函数）

- **input size *n***: *number of items in the input*, depends on the problem
  - Sorting array A: n = length of A
  - Graph: n can be the numbers of verices V and edges E
  - Multiplication: total number of bits
  - etc.
- **Running time $T(n)$**: on a particular input, *the number of primitive operations or "steps" executed* （计算时间，或运行时间：针对某个输入，算法执行的基本操作数）
- **principle of consistency Running time** （运行时间满足如下一致性原则）
  - The steps are machine-independent（基本操作独立于机器）
  - A constant amount of time c_i is required to execute the *i*th line pseudocode（每行指令的执行时间为常数）

2.2.1 Analysis of insertion sort

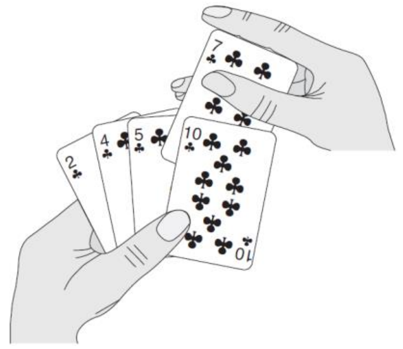
```
INSERTION-SORT(A)                                              cost      times
1     for( j = 2; j <=length[A]; j++)                          $c_1$     $n$
2     {     key = A[j]                                         $c_2$     $n-1$
3           // Insert A[j] into the sorted sequence A[1 .. j-1]  0       $n-1$
4           i = j-1                                            $c_4$     $n-1$
5           while( i > 0 && 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           }
9           A[i+1] = key                                       $c_8$     $n-1$
10    }
```





$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Analysis of insertion sort



```
INSERTION-SORT(A)
1  for( j = 2; j <=length[A]; j++) // loop header
2  {       key = A[j]
3          // Insert A[j] into the sorted sequence
4          i = j-1
5          while( i > 0 && A[i] > key)
6          {     A[i+1] = A[i]
7                i = i-1
8          }
9          A[i+1] = key
10 }       // loop body below
```

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

Even for inputs of a given size, an algorithm’s running time may depend on which input of that size is given.（相同输入规模的情况下，输入不同，运算时间也可能不同）

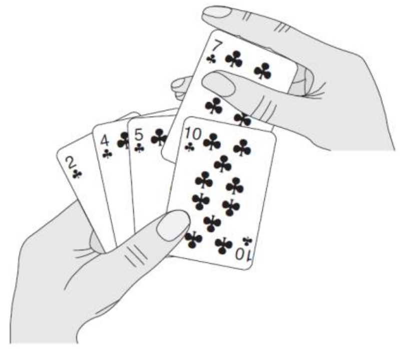
The best case occurs if the input is already sorted. For each $j = 2, 3, \ldots, n$, $A[i] \le key$ in line 5 when i has its initial value of j-1. Thus $t_j = 1$, and the best-case running time is

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) = an + b$$

It is thus a **linear function** of *n*.

Analysis of insertion sort



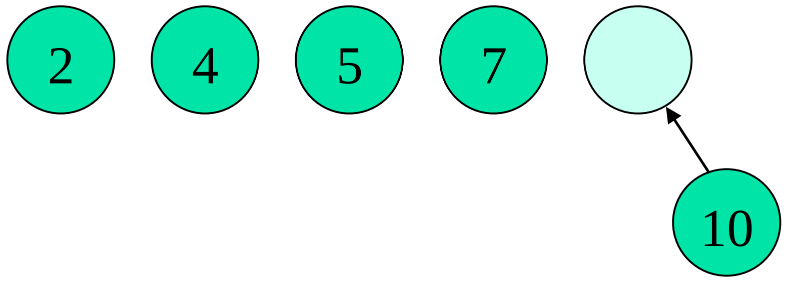
```
INSERTION-SORT(A)
1  for( j = 2; j <=length[A]; j++) // loop header
2  {        key = A[j]
3           // Insert A[j] into the sorted sequence
4           i = j-1
5           while( i > 0 && A[i] > key)
6           {     A[i+1] = A[i]
7                 i = i-1
8           }
9           A[i+1] = key
10 }       // loop body below
```

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

Even for inputs of a given size, an algorithm’s running time may depend on which input of that size is given.（相同输入规模的情况下，输入不同，运算时间也可能不同）

best case: $T(n) = an + b$

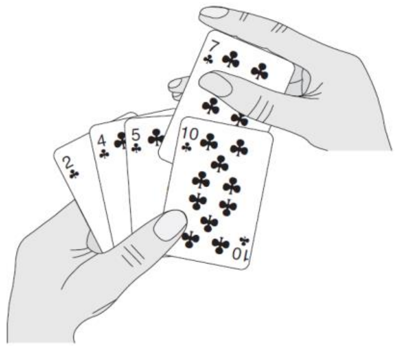
仅考虑第5行的 $A[i] > key$ 作为基本操作，每次for循环， $A[i] > key$ 仅执行一次（因为条件不成立，while循环体不执行），算法共执行 n-1 次比较。



key:

$A[5] = 10$

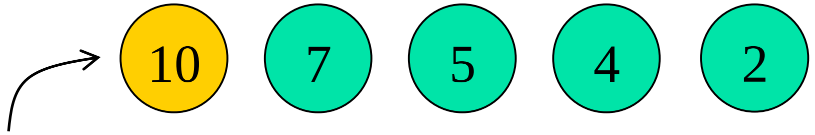
Analysis of insertion sort



```
INSERTION-SORT(A)
for( $j$ = 2; $j$ <=length[A]; j++) // loop header
{       key = A[j]
        // Insert A[j] into the sorted sequence
        i = j-1
        while( i > 0 && A[i] > key)
        {    A[i+1] = A[i]
             i = i-1
        }
        A[i+1] = key
}      // loop body below
```

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

An algorithm's running time may depend on factual input.



The worst case results if the input is in reverse sorted order（输入为逆序）. We must compare each element $A[j]$ with each element in the entire sorted subarray $A[1 .. j-1]$, and so $t_j = j$. Noting that

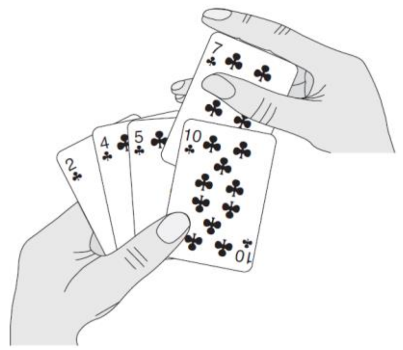
$$\sum_{j=2}^{n} j = (\sum_{j=1}^{n} j) - 1 = \frac{n(n+1)}{2} - 1, \qquad \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

Then

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(\frac{n(n+1)}{2} - 1) + c_6(\frac{n(n-1)}{2}) + c_7(\frac{n(n-1)}{2}) + c_8(n-1) = an^2 + bn + c$$

It is thus a **quadratic function (n^2)** of n .

Analysis of insertion sort



```
INSERTION-SORT(A)
1  for( j = 2; j <=length[A]; j++) // loop header
2  {      key = A[j]
3         // Insert A[j] into the sorted sequence
4         i = j-1
5         while( i > 0 && A[i] > key)
6         {     A[i+1] = A[i]
7               i = i-1
8         }
9         A[i+1] = key
10 }     // loop body below
```

$$T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n}t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8(n-1)$$

An algorithm’s running time may depend on factual input.

Exercise:

worst case: $T(n) = an^2 + bn + c$

$$\sum_{j=2}^{n} j=(\sum_{j=1}^{n} j)-1=\frac{n(n+1)}{2}-1$$

仅考虑第 5 行的条件作为基本操作，算法共执行？次基本操作。

2.2.1 Analysis of insertion sort

best case

$$T(n) = an + b$$

worst case

$$T(n) = an^2 + bn + c$$

```
INSERTION-SORT(A)
for( $j$ = 2; $j$ <= length[A]; $j$++) // loop header
{   key = A[j]
    // Insert A[j] into the sorted sequence
    $i$ = $j$-1
    while( $i > 0$ && A[i] > key)
    {   A[i+1] = A[i]
        $i$ = $i$-1
    }
    A[i+1] = key
} // loop body below
```

插入排序的效率实际非常高！
对于“好”的输入，比快排还快很多！

2.2.2 Worse-case and average-case analysis

- **Worst-case: the longest running time for any input of size *n*.**
- **We usually concentrate on finding the worst-case.**
  - The worst-case running time of an algorithm is an upper bound on the running time for any input.
  - The worst case occurs fairly often. (in searching a databse)
    （最坏情况经常发生）
  - The "average case" is often roughly as bad as the worst case. (suppose that we randomly choose *n* numbers and apply insertion sort.)

因此，OJ 上测试时，样例通过不算，记得测试各种边界条件，如：题目中说明输入n个数($1 \le n \le 10^6$)，但样例只给了5个数，程序需要考虑数据为10^6的情形，且这10^6个数的各种类型的顺序都应考虑！

- **Average-case: or expected running time of an algorithm**
  - Probabilistic analysis (Knuth)
  - Randomized algorithm

2.2.3 Order of growth

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(\frac{n(n+1)}{2} - 1) + c_6(\frac{n(n-1)}{2}) + c_7(\frac{n(n-1)}{2}) + c_8(n-1) = an^2 + bn + c = \Theta(n^2) \approx n^2$$

Rate of growth (order of growth，函数增长率)

- the more simplifying abstraction.（函数更抽象、简洁的表示）
- considering only the leading term of a formula (e.g., $\boldsymbol{an^2}$ of $\boldsymbol{an^2+bn+c}$), since the lower-order terms are relatively insignificant for large n.（**抓主要矛盾**）
- ignoring the leading term's constant coefficient, since constant factors are less significant than the rate of growth in determining computational efficiency for large inputs. （忽略高阶项的常系数）
- Thus, the insertion sort has a worst-case running time of $\Theta(n^2)$.
- Usually, one algorithm is more efficient than another if its worst-case running time has a lower order of growth (small inputs exception, e.g., $100n\lg n$ vs $2n^2$).

2.2.3 Order of growth

Rate of growth (order of growth，函数增长率)

$T_1(n) = 100n\lg n < T_2(n) = 2n^2$ （ n 很大时的含义 ）

数学与计算机算法的关系

- 数学看上去很美！计算机算法不但美，还很真实！
- 数学靠想像，猜想，靠推理，算法能够用程序实现，能计算出结果。
- 数学更多是研究无穷的情况，算法研究有限的情况。
- 数学的无穷远处能想像，很美，但看不见够不着，算法很大的地方能想像，也很美，看得见但原始人力够不着的时候计算机可以够得着。
- 数学帮助算法想像计算世界的美，算法帮助数学实现和走进计算世界的美。

Exercise

编程实现该插入排序算法。产生规模为10^5的整数作为输入数据，用你的程序对数据进行排序。数据至少需要包括如下3组：

1. 正序数据；
2. 随机序数据；
3. 逆序数据。

对于每组数据，统计基本操作（第6行）执行的次数。结果说明了什么？

```
INSERTION-SORT(A)
1    for( $j$ = 2; $j$ <= $length[A]$; $j$++)
2    {
3        $key = A[j]$
4        // Insert $A[j]$ into thesequence
5        $i = j-1$
6        while( $i > 0$ && $A[i] > key$)
7         {
8            $A[i+1] = A[i]$
9            $i = i-1$
10        }
11       $A[i+1] = key$
12   }
```

强调：写好伪代码是写好代码的关键！

将来要提交的伪代码和代码应该这样写！

```
INSERTION-SORT(A)
for( $j = 2$; $j <= length[A]$; $j$++)
{
    $key = A[j]$
    $i = j-1$
    while( $i > 0$ && $A[i] > key$)
    {
        $A[i+1] = A[i]$
        $i = i-1$
    }
    $A[i+1] = key$
}
```

```
void insert_sort(int x[], int n)
{
    int key, i, j;
    for(j=1; j<n; j++)  //数组下标从0开始
    {
        key = x[j];
        i = j-1;
        while(i>=0 && x[i]>key)
        {
            x[i+1] = x[i];
            i--;
        }
        x[++i] = key;
    }
}
```

本讲小结

1. 课程安排
2. 算法是什么、学习算法的意义
3. 算法的描述：自然语言、流程图、伪代码、编程语言
4. 算法的正确性证明：循环不变量
5. 算法分析的理论：计算模型、基本步骤、复杂度度量
6. 用实例 insertSort 阐述了算法描述、算法正确性证明、算法复杂度分析