计算的代价和局限

—— 什么是难问题？不可解问题？

2021.11.16

任重道远

计算的代价

我们关心的问题

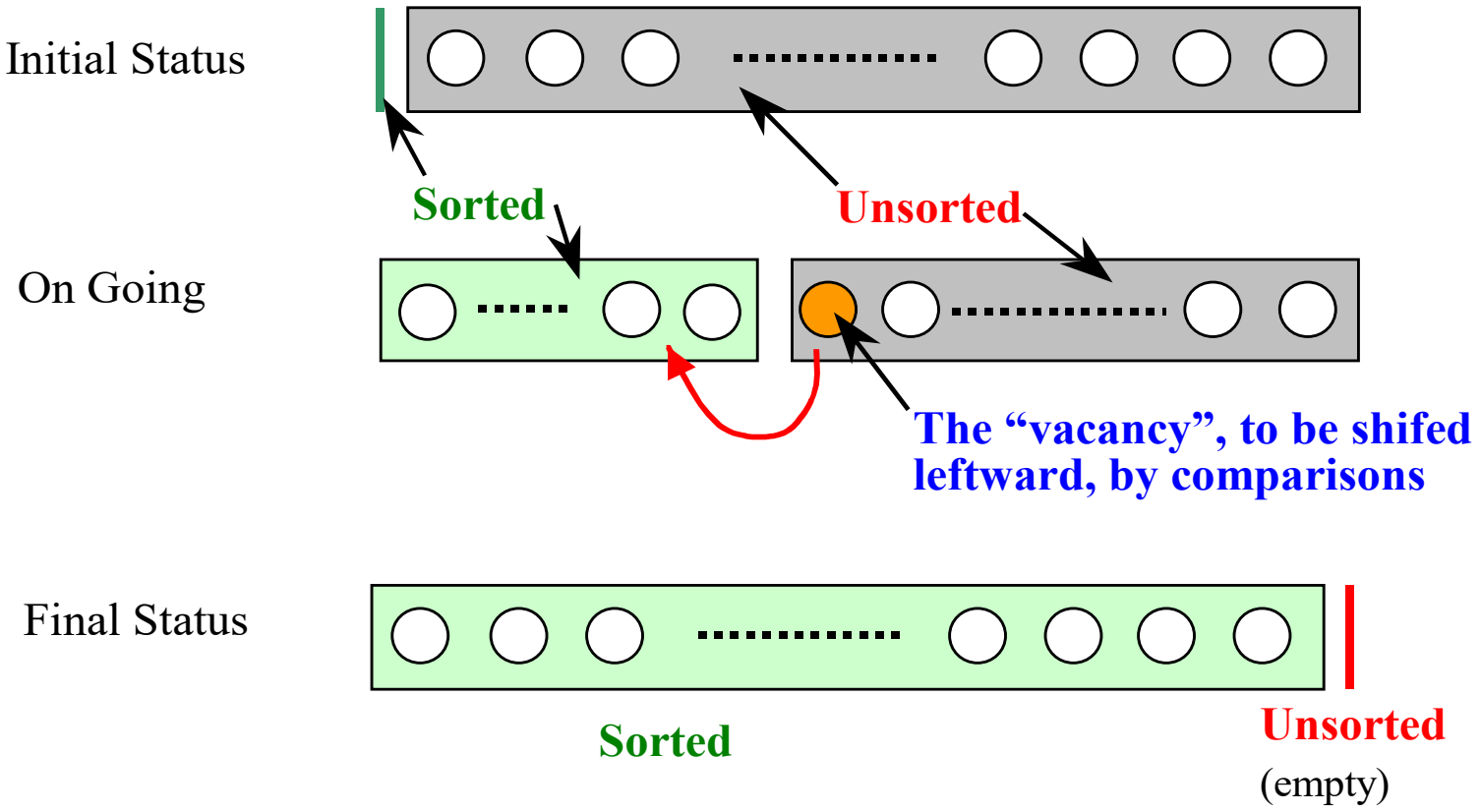
- 问题该如何解?
- 为什么这个解法是正确的?
- 最坏情况下需要多少代价?
- 对任意输入实例求解的代价平均期望值是多少?
- 这个解法还可以改进吗?

计算需要多少代价?

- 代价?
  - 时间
  - 空间

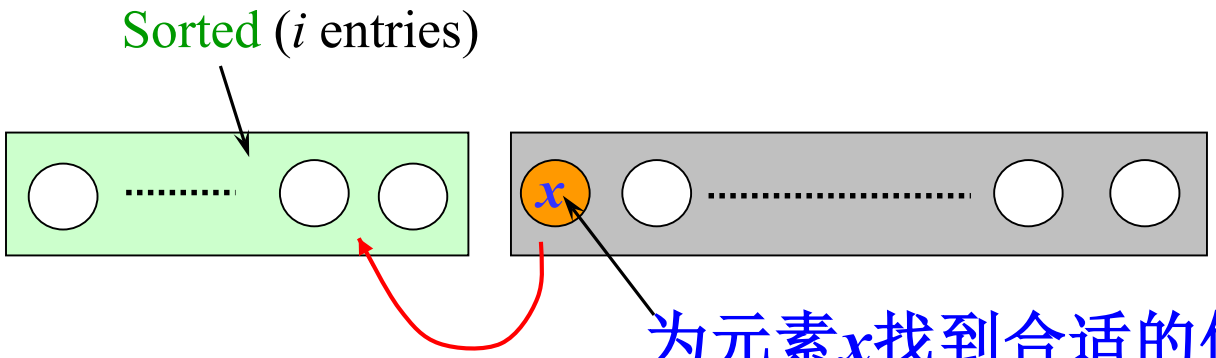
- 求解一个问题可能有很多种方法
- 计算代价的差异
  - 同一种方法不同输入的计算代价不同
  - 同一个输入不同方法的计算代价不同

插入排序



注意：总是比较相邻的元素

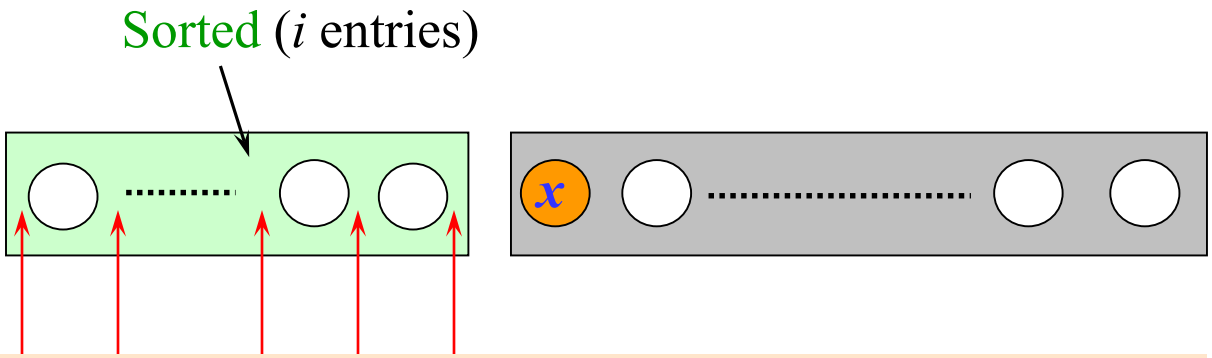
最坏情况的代价



- 最初，未排序部分有（n-1）个记录，于是

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

平均行为



x 可能出现在(i+1)的空位中的任何一个，并且可以认为在任何一个空位的概率是相同的。

- 假设：
  - 输入序列中的各种排列方式概率相同。
  - 同一个关键字只有一个入口点。

注意：对于第(i+1)空位，即最左边的空位，只需要i次比较即可。

平均计算代价

- 对于第i个元素，要找到它的位置，需要进行这么多次的比较：

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

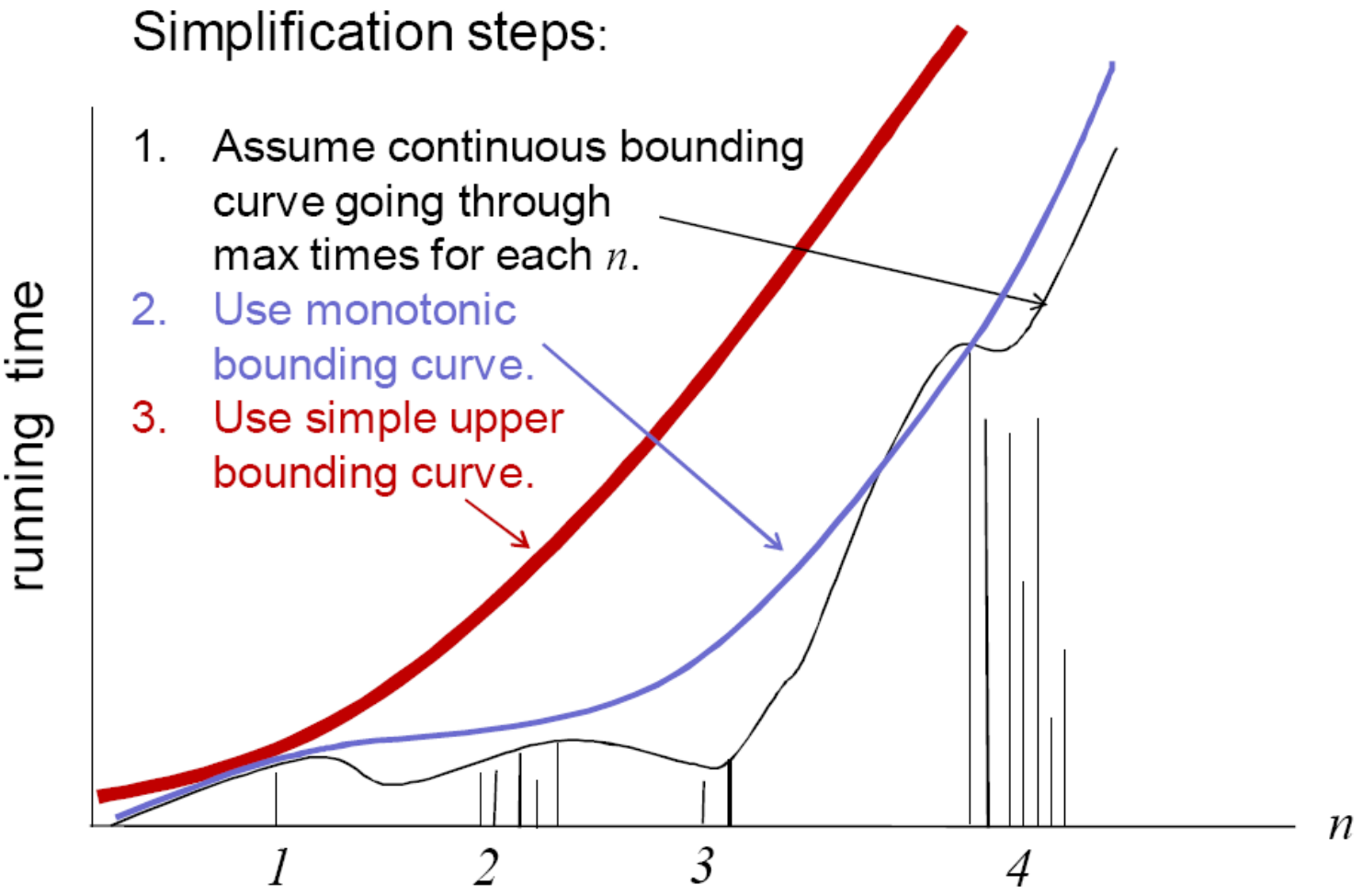
最左边的空位

- 对于所有n-1 次插入。

$$A(n) = \sum_{i=1}^{n-1}\left(\frac{i}{2} + 1 - \frac{1}{i+1}\right) = \frac{n(n-1)}{4} + n - 1 - \sum_{j=2}^{n}\frac{1}{j}$$

$$= \frac{n(n-1)}{4} + n - \sum_{j=1}^{n}\frac{1}{j} = \frac{n^2}{4} + \frac{3n}{4} + \ln n$$

用一个“简单”函数来表示

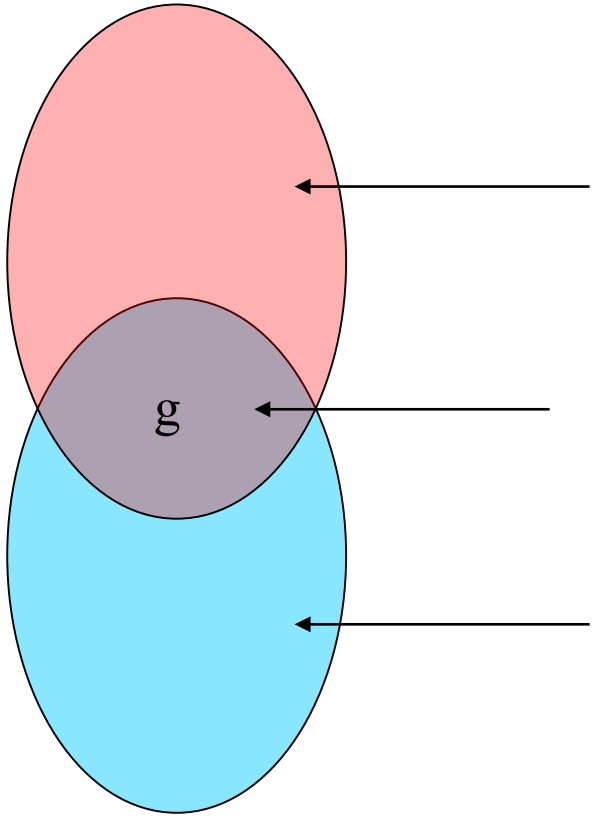

Simplification steps:
1. Assume continuous bounding curve going through max times for each n.
2. Use monotonic bounding curve.
3. Use simple upper bounding curve.
running time
n
1
2
3
4

渐进行为

- 分析算法代价的重要手段
- 表示方式：O、Θ、Ω

- 假设总的执行步骤与基本的操作是成比例的
- 仅仅考虑影响最大的项
- 忽略项前系数

相应增长率



Ω(g):functions that grow at least as fast as g

Θ(g):functions that grow at the same rate as g

O(g):functions that grow no faster as g

The Set "Big Oh"

- Definition
  - Giving g:N$\rightarrow$R$^+$, then O(g) is the set of f:N$\rightarrow$R$^+$, such that for some c$\in$R$^+$ and some n$_0\in$N, $f(n)\leq cg(n)$ for all $n \geq n_0$.
- A function $f\in$O(g) if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$
  - Note: c may be zero. In that case, $f\in$o(g), "little Oh"

Example

Using L'Hopital's Rule

- Let $f(n)=n^2$, $g(n)=n\lg n$, then:
  - $f \notin O(g)$, since

$$\lim_{n\to\infty}\frac{n^2}{n\lg n}=\lim_{n\to\infty}\frac{n}{\lg n}=\lim_{n\to\infty}\frac{n}{\frac{\ln n}{\ln 2}}=\lim_{n\to\infty}\frac{1}{\frac{1}{n\ln 2}}=\infty$$

  - $g \in O(f)$, since

$$\lim_{n\to\infty}\frac{n\log n}{n^2}=\lim_{n\to\infty}\frac{\log n}{n}=\lim_{n\to\infty}\frac{\ln n}{n\ln 2}=\lim_{n\to\infty}\frac{1}{n\ln 2}=0$$

For your reference: L'Hôpital's rule

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

with some constraints

插入排序的渐进复杂度

- 最坏情况

$$W(n) \le \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$$

- 平均情况

$$A(n) = \frac{n^2}{4} + \frac{3n}{4} + \ln n \in \Theta(n^2)$$

逆序对与排序

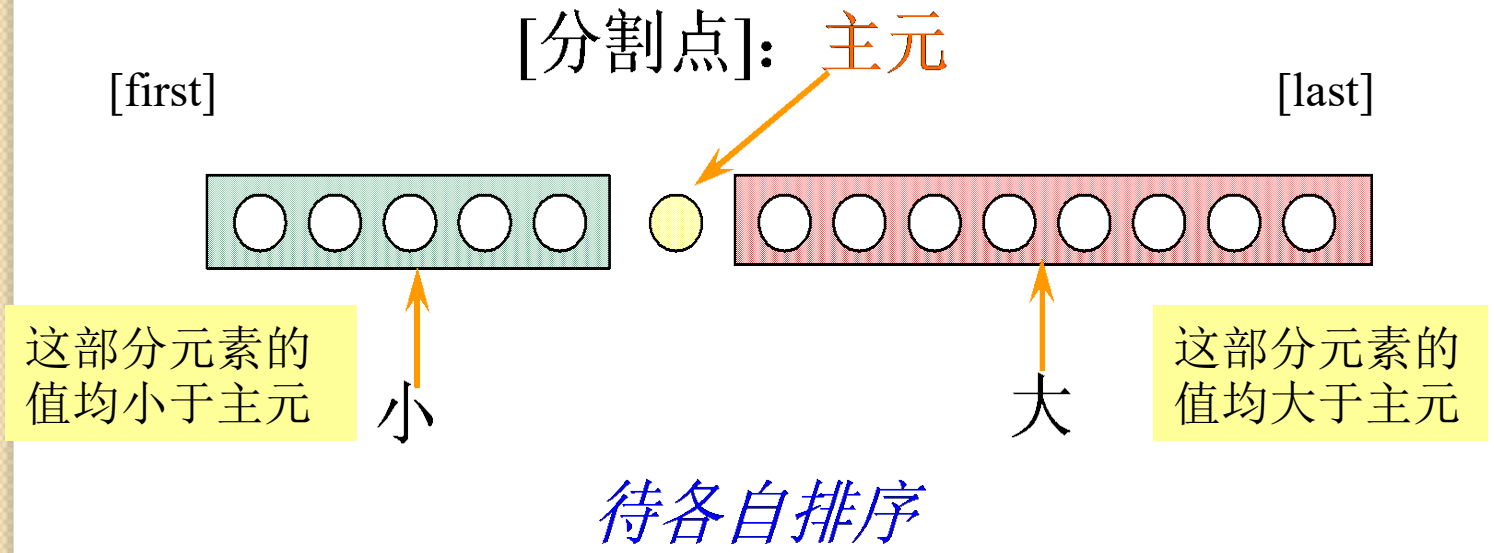
- 未排序序列 *E*:

$$x_1, x_2, x_3, \ldots, x_{n-1}, x_n$$

- 如果没有相同的关键字，可以将任意的关键字排序视为自然数排序。
  - $\{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}=\{1,2,3,\ldots,n-1,n\}$
- 如果$x_i>x_j$，但是 $i<j$，则 $<x_i, x_j>$ 是逆序对。
- 排序过程必须要消除所有的逆序对。

改进的解法：一步跳很远

- 将待排序的元素划分为“大”和“小”的两个部分，这两部分将各自再排序。



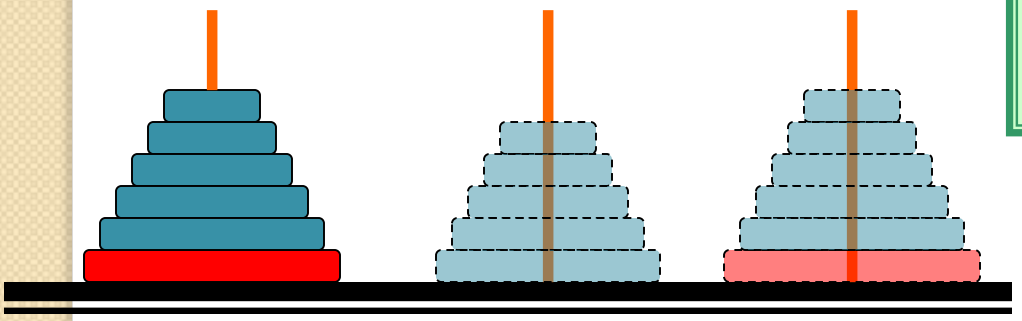
$O(n\log n)$

双拳难敌四手

P、NP、NPC

Hanoi问题

- Towers of Hanoi
  - How many moves are need to move all the disks to the third peg by moving only one at a time and never placing a disk on top of a smaller one.



$$T(1) = 1$$
$$T(n) = 2T(n\text{-}1) + 1$$

Hanoi问题的复杂度

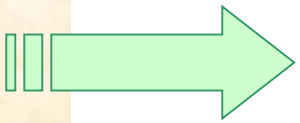
$T(n) = 2T(n-1) + 1$

$2T(n-1) = 4T(n-2) + 2$

$4T(n-2) = 8T(n-3) + 4$

……

$2^{n-2}T(2)$



$T(n) = 2^n - 1$

It is **extremely difficult** to achieve the result for an input of moderate size. For the input of 64, it takes half a million years even if the Tibetan priest has superhuman strength to move a million discs in a second.

时间复杂度的增长

Algorithm	1	2	3	4	5
Time function(ms)	$33n$	$46n \lg n$	$13n^2$	$3.4n^3$	2^n
Input size(n)	Solution time				
10	0.00033 sec.	0.0015 sec.	0.0013 sec.	0.0034 sec.	0.001 sec.
100	0.0033 sec.	0.03 sec.	0.13 sec.	3.4 sec.	4×10^{16} yr.
1,000	0.033 sec.	0.45 sec.	13 sec.	0.94 hr.	
10,000	0.33 sec.	6.1 sec.	22 min.	39 days	
100,000	3.3 sec.	1.3 min.	1.5 days	108 yr.	
Time allowed	Maximum solvable input size (approx.)				
1 second	30,000	2,000	280	67	20
1 minute	1,800,000	82,000	2,200	260	26

最大团问题

- 一个图中包含的完全子图称为一个团（clique），团的大小就是这个子图包含的节点数目。
  - 优化问题：给定一个图G，找到G的最大团
  - 判定问题：给定一个图G和一个常数k，G的最大团的大小是否至少为k?

优化与判定

- 一般而言，一个优化问题都可以转化为一个判定问题。
- 对于一些问题，可以证明，判定能在多项式时间完成，当且仅当对应的优化问题能在多项式时间内完成。
- 我们可以断言，如果一个判定问题不能在多项式时间内解答，那么对应的优化问题也不可能。

再次分析最大团问题

- 最大团问题（找到图G的最大团）能在多项式时间内完成，当且仅当对应的判定问题能在多项式时间内完成。
  - 如果一个图G的最大团能够在$g(n)$的时间内找到，那么当然可以在这样的时间内给出相应判定问题的解。
  - 如果判定问题能够在$f(n)$时间内完成，那么我们只需要运用n次这样的判定方法，就必然能找到最大团，于是时间复杂度为$nf(n)$，同样也是多项式时间。

*P*类问题

- 多项式（界）算法
  - polynomially bounded algorithm
  - 最坏情况下算法复杂度是其输入规模的一个多项式函数
- 多项式（界）问题
  - polynomially bounded problem
  - 存在多项式算法能解答的问题
- *P*类问题就是多项式（界）判定问题的集合。

非确定性算法

```
void nondetA(String input)
    String s=genCertif();
    Boolean CheckOK=verifyA(input,s);
    if (checkOK)
        Output “yes”;
    return;
```

阶段1 猜想：产生任意的推荐解答

阶段2 验证：确定s是否是一个有效的回答，并且是一个满足要求的解答。

算法可能会对同样的输入产生不同的执行方式：“是”或者没有输出。

非确定性 vs. 确定性

在*n*个元素中找到一个特定的元素

```
void nondetSearch(int k; int[ ] S)
    int i =genCertif();
    if (S[i]=k)
        Output "yes";
    return;
```

***O*(1)**

注意：对应的确定性算法是$\Omega(n)$的复杂度

对*n*个元素进行排序

```
void nondetSort(int[ ] S; int n)
    int i, j; int[ ] out=0;
    for i =1 to n do
        j= genCertif();
        if out[j ]≠0 then return;
        out[j]=S[i ];
    for i =1 to n-1 do
        if out[i ]>out[i+1] then return;
    S=out;
    Output(yes);
return
```

***O*(*n*)**

注意：对应的确定性算法是$\Omega(n\log n)$的复杂度

*NP*类问题

- 多项式（界）非确定性算法
  - polynomial bounded nondeterministic algorithm
  - 对于每一个规模为n，且应当产生“是”输出的输入，算法能在多项式时间给出“是”的输出。
- *NP*类问题是具有多项式（界）非确定性算法的判定问题的集合。

最大团问题是NP问题

```
void nondeteClique(graph G; int n, k)
    set S=ϕ;
    for int i=1 to k do
        int t=genCertif();
        if t∈S then return;
        S=S∪{t };
    for all pairs (i,j) with i,j in S and i≠j do
        if (i,j ) is not an edge of G
            then return;
        Output("yes");
```

$O(n)$

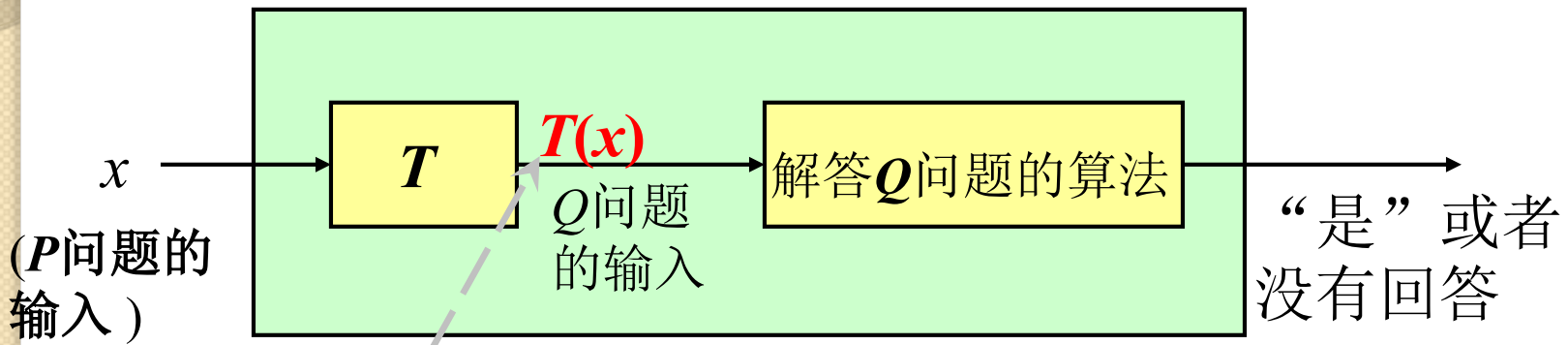
$O(k^2)$

所以，我们可以得到一个时间复杂度为$O(n+k^2)=O(n^2)$的非确定性算法。

*P*和*NP*的关系

- 对于判定问题的一个确定性算法可以认为是一个非确定性算法的特例，也就意味着$P \subseteq NP$
  - 确定性算法可以认为是非确定性算法的第2阶段，并忽略第1阶段
- 直观而言，*NP*比*P*的集合大很多。
  - 猜想的可能解答有指数多个
  - 没有任何一个*NP*类问题被证明不属于*P*类

间接性解答问题



对P问题，输入为x的正确回答为“是”，当且仅当对Q问题，输入$T(x)$的正确回答为“是”。

如果T是多项式时间内完成，我们认为P可以多项式时间归约到Q，记为$P \leq_P Q$

NP完全问题

- 如果**任何**属于NP类的问题都可以规约为Q，即$P \leq_P Q$，那么这个问题Q就称为***NP难***问题。
  - Q至少与NP类中任何问题一样难。

- 如果问题Q是一个NP类问题，同时也是NP难问题，那么Q称为***NP完全***问题。
  - Q最多还是能被多项式（界）的非确定性算法解答。

一个*NP*难问题

- 停机问题：给定一个任意的确定性算法A和输入I，是否会停机呢?

- 著名的不可判定问题，显然不是*NP*类问题。

*P*和*NP*

- 直观而言，*NP*比*P*的集合大很多。
  - 猜想的可能解答有指数多个。
  - 没有任何一个*NP*类问题被证明不属于*P*类。

- 如果任何一个*NP*完全问题属于*P*类，那么*NP*=*P*

qzz@nju.edu.cn

钱柱中