1.绪论

(x2) 排序与下界

两个吃罢饭，又走了四五十里，却来到一市镇上，地名唤做瑞龙镇，却是个三岔路口。宋江借问那里人道：“小人们欲投二龙山、清风镇上，不知从那条路去？”

邓俊辉

deng@tsinghua.edu.cn

难度与下界

- 由前述实例可见，同一问题的不同算法，复杂度可能相差悬殊
- 在可解的前提下，可否谈论问题的难度？如何比较不同问题的难度？
- 问题P若存在算法，则所有算法中**最低**的复杂度称为P的难度
- 为什么要确定问题的难度？给定问题P，如何确定其难度？
- 两个方面着手：

  设计复杂度更低的算法 ＋ 证明更高的问题难度下界
- 一旦算法的复杂度达到难度下界，则说明

  就大O记号的意义而言，算法已经最优
- 例如，排序问题下界为$\Omega(n\log n)$，而且是紧的...

排序

- 任给n个元素$\{R_1, \ldots, R_n\}$，对应关键码$\{K_1, \ldots, K_n\}$
  需按某种次序排列 //广义的次序：≤，偏序/全序
- 亦即，找出< 1, ..., n >的一个**排列**< $i_1, \ldots, i_n$ >，使得 $K_{i1} \le \ldots \le K_{in}$
  { 3, 1, 4, 1, 5, 9, 2, 6 } → { 1, 1, 2, 3, 4, 5, 6, 9 }
- 注意：此处的关键码集是复集（multiset） //可能存在重复关键码
- 应用：25~50%的计算都可归于排序
  飞机调度系统：(航班号, 起降时间)
  文件系统：(文件名, 扩展名, 大小, 日期)
  加速查找：电话簿
  加速比对：在花名册A和B之间，找出重复出现者
  ...

算法分类

❖ 直接算法：直接移动元素本身 //元素结构简单时适宜采用

❖ 间接算法：下标 + 关键码 + 指针 //元素结构复杂时适宜采用

❖ 内部 / 外部 (internal / external)

❖ 脱机 / 在线 (offline / online)

❖ 串行 / 并行 (sequential / parallel)

❖ 确定性 / 随机 (deterministic / randomized)

❖ 基于比较式 / 散列式 (comparison-based / hash-based)

时空性能、稳定性

❖ 多种角度估算的时间、空间复杂度

最好 / best-case　　最坏 / worst-case

平均 / average-case　　分摊 / amortized

❖ 其中，对最坏情况的估计最保守、最稳妥

因此，首先应考虑**最坏**情况**最优**（worst-case optimal）的算法

❖ 排序所需的时间，主要取决于

关键码比较的次数 / #key comparisons

元素交换的次数 / #data swaps

❖ 就地（in-place）：除输入数据本身外，只需$\mathcal{O}(1)$附加空间

❖ 退化（degeneracy）：若存在雷同关键码，则排序结果不唯一

❖ 稳定（stable）算法：关键码雷同的元素，在排序后相对位置保持

最坏情况最优 + 基于比较

❖ 排序算法，最快能够有多快？

语境1：就最坏情况最优而言

语境2：就某一大类主流算法而言...

❖ 基于比较的算法（comparison-based algorithm）

算法执行的进程，取决于一系列的数值（这里即关键码）比对结果

比如，max() 和 bubbleSort()

❖ 任何CBA在最坏情况下，都需$\Omega(n\log n)$时间才能完成排序

判定树

❖ 每个CBA算法，都对应于一棵判定树

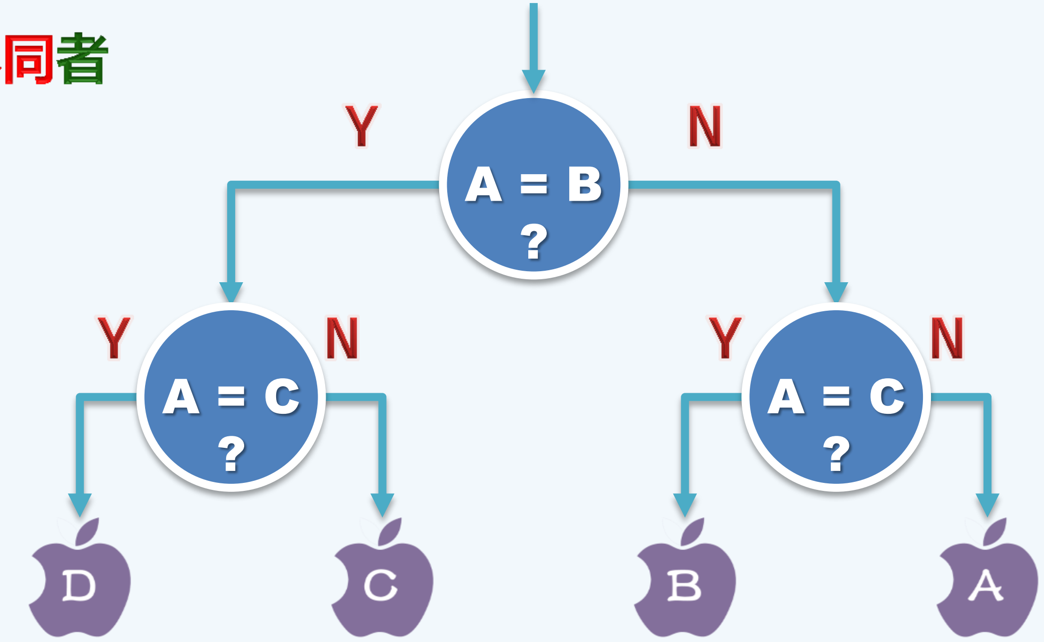
从根节点通往任一叶子的路径，都对应于算法的某次运行过程

每一可能的输出，都对应于至少一匹叶子（一条通往叶子的路径）

❖ 实例： 经过2/4次称量

必可从4/16只苹果中，找出唯一的重量不同者

❖ 问题： 称量次数可否更少？



代数判定树

❖ 代数判定树 (algebraic decision tree)

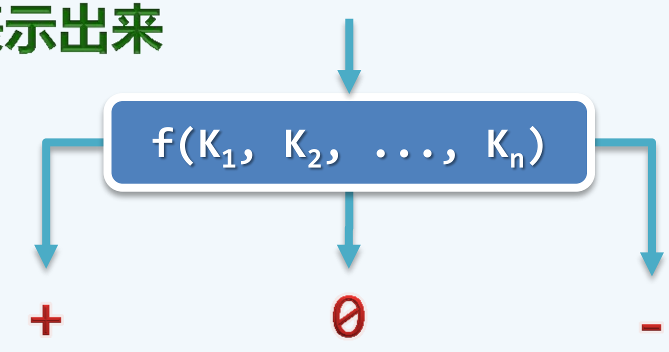
针对“比较-判定”式算法的计算模型

给定输入的规模，将所有可能的输入所对应的一系列判断表示出来

代数判定：　使用某一常次数代数多项式

将任意一组关键码做为变量，对多项式求值

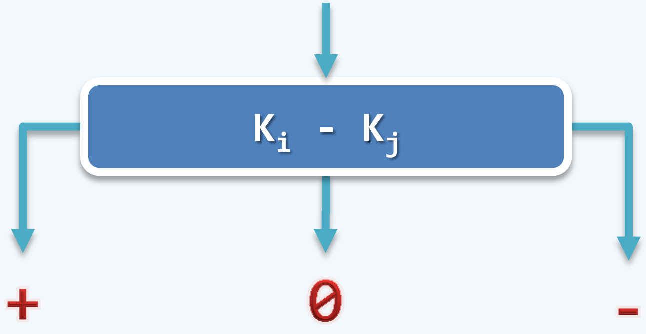
根据结果的符号，确定算法推进方向



❖ 比较树 (comparison tree)

最简单的ADT

二元一次多项式，形如：$K_i - K_j$



下界：Ω(nlogn)

- 比较树是三叉树（ternary tree）
  - 每个内节点至多三个分支 //+、0、-
  - 从根节点到叶子的每一路径，对应于算法的某次运行过程
  - 每匹叶子对应于一个输出 //在此，即排序后的序列
  - 树高 = 最坏情况下所需的比较次数 //最坏情况最优
  - 树高的下界 = 所有CBA的时间复杂度下界
- 对n个元素进行排序的任何一棵ADT，高度至少为Ω(nlogn)
  - #叶子 ≥ #可能的输出 = n个元素可能的排列 = n!
  - 树高 $\geq log_3 n! = log_3 e \cdot ln(n!)$

    $= log_3 e \cdot [nlnn - n + O(lnn)] = \Omega(nlogn)$ //Stirling approximation
- 上述结论，可进一步推广至理想平均情况、随机情况（概率≥25%）...

熵与下界

❖ 热力学第二定律：能量不会自动地从低温物体传向高温物体

❖ Shannon： 数据系统S中蕴含的信息量，可由信息熵度量

Entropy(S) ~ $\log_2 N$ （N = S可能的状态总数）

C. E. Shannon
(1916 – 2001)

❖ 热系统的熵减少，都须付出一定的能量

数据系统的信息熵减少，也须付出一定的计算量

? ? ? ? ? ? ?

$E(S_0) \sim \log_2(n) = \Omega(\log n)$

鉴别

x

$E(S_1) \sim \log_2 1 = 0$

2 5 7 3 4 1 6

$E(S_0) \sim \log_2(n!) = \Omega(n\log n)$

排序

1 2 3 4 5 6 7

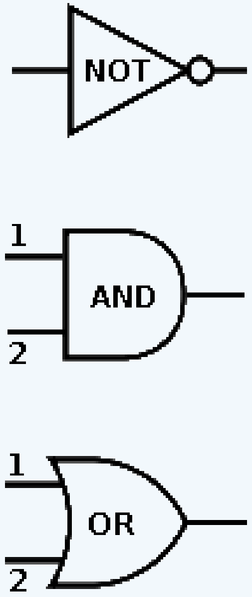
$E(S_1) \sim \log_2 1 = 0$

熵与下界

❖ Landauer's principle

信息的减少或丢失，必伴随着熵的增加，并发出相等的热量

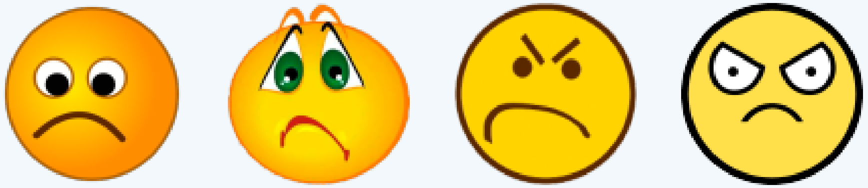
就最低功耗而言，AND和OR逻辑门必高于NOT门



❖ “幸福的人都是一样的，不幸福的人却各有各的不幸”

从计算的角度看，幸福的可能状态，要远少于不幸

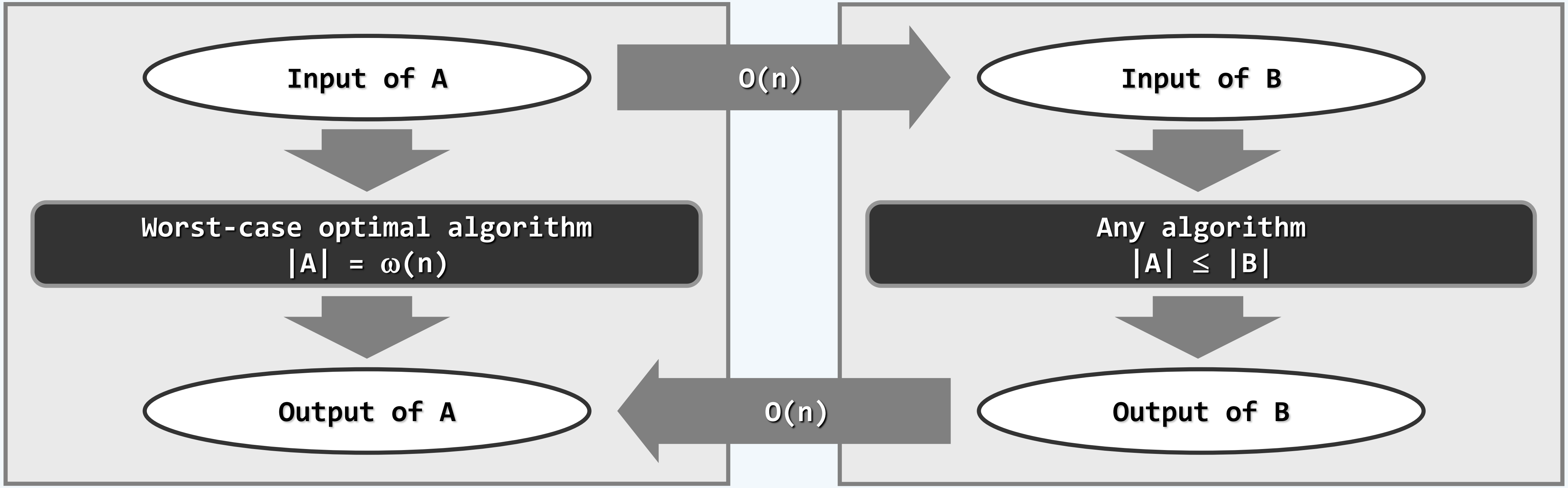
前者的熵远小于后者，故从后者到前者，需要付出巨大的努力



规约：线性规约

❖ 除了（代数）判定树，规约（reduction）也是确定下界的有力工具

$\mathcal{O}$(nlogn)	linear-time reduction
NP-complete/P	polynomial-time reduction
P-SPACE complete	polynomial-time many-one reduction



规约：实例

- Sorting $\leq_N$ Red-Blue Matching
- 【Element Uniqueness】任意n个实数中，是否包含雷同？
  EU的下界为$\Omega(n\log n)$
- EU $\leq_N$ Closest Pair
- 【Integer Element Uniqueness】任意n个整数中，是否包含雷同？
  IEU的下界为$\Omega(n\log n)$！
- IEU $\leq_N$ Segment Intersection Detection
- 【Set Disjointness】任意一对集合A和B，是否存在公共元素？
  SD的下界为$\Omega(n\log n)$
- SD $\leq_N$ Diameter