Lower Bound Arguments

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。
- 一个问题的下界不是唯一的，且下界越高越好。

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。
- 一个问题的下界不是唯一的，且下界越高越好。

例如：**$\Omega(1)$, $\Omega(n)$ and $\Omega(n\log n)$** 均是排序问题的下界，显然 **$\Omega(n\log n)$** 是最好的一个下界.

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。
- 一个问题的下界不是唯一的，且下界越高越好。

例如：$\Omega(1)$, $\Omega(n)$ and $\Omega(n\log n)$ 均是排序问题的下界，显然 $\Omega(n\log n)$ 是最好的一个下界.

- 已知一个问题目前的最好下界为$\Omega(n)$ ，且最好的算法的时间复杂度为$O(n^2)$，则

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。
- 一个问题的下界不是唯一的，且下界越高越好。

例如：$\Omega(1)$, $\Omega(n)$ and $\Omega(n\log n)$ 均是排序问题的下界，显然 $\Omega(n\log n)$ 是最好的一个下界.

- 已知一个问题目前的最好下界为$\Omega(n)$ ，且最好的算法的时间复杂度为$O(n^2)$，则
  - 可以尝试找到一个更好的下界，如 $\Omega(n\log n)$

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。
- 一个问题的下界不是唯一的，且下界越高越好。

例如：$\Omega(1)$, $\Omega(n)$ and $\Omega(n\log n)$ 均是排序问题的下界，显然 $\Omega(n\log n)$ 是最好的一个下界.

- 已知一个问题目前的最好下界为$\Omega(n)$ ，且最好的算法的时间复杂度为$O(n^2)$，则
  - 可以尝试找到一个更好的下界，如 $\Omega(n\log n)$
  - 可以尝试找到一个更好的算法，如$O(n\log n)$时间算法

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。
- 一个问题的下界不是唯一的，且下界越高越好。

例如：$\Omega(1)$, $\Omega(n)$ and $\Omega(n\log n)$ 均是排序问题的下界，显然 $\Omega(n\log n)$ 是最好的一个下界.

- 已知一个问题目前的最好下界为$\Omega(n)$ ，且最好的算法的时间复杂度为$O(n^2)$，则
  - 可以尝试找到一个更好的下界，如 $\Omega(n\log n)$
  - 可以尝试找到一个更好的算法，如$O(n\log n)$时间算法
  - 或者以上可同时达到

下界

- 一个问题的下界是指解决这个问题的任何算法所需的最小的计算量。
- 下界一般通过理论的推导得到。
- 一个问题的下界不是唯一的，且下界越高越好。

例如：$\Omega(1)$, $\Omega(n)$ and $\Omega(n\log n)$ 均是排序问题的下界，显然 $\Omega(n\log n)$ 是最好的一个下界.

- 已知一个问题目前的最好下界为$\Omega(n)$ ，且最好的算法的时间复杂度为$O(n^2)$，则
  - 可以尝试找到一个更好的下界，如 $\Omega(n\log n)$
  - 可以尝试找到一个更好的算法，如$O(n\log n)$时间算法
  - 或者以上可同时达到
- 如果一个问题目前的下界为$\Omega(n\log n)$，且存在一个 $O(n\log n)$时间的算法，则可认为该算法为最优的

下界

- 平凡下界（**Trivial lower bound**）
- 信息论论证（**Information-Theoretic Arguments**）
- 对手论证 （**Adversary Arguments**）
- 问题归约（**Problem Reduction**）
  - 线性时间归约

下界

- **平凡下界（Trivial lower bound）**
- **信息论论证（Information-Theoretic Arguments）**
- **对手论证 （Adversary Arguments）**
- **问题归约（Problem Reduction）**

下界

- **平凡下界（Trivial lower bound）**
- **信息论论证（Information-Theoretic Arguments）**
- **对手论证 （Adversary Arguments）**
- **问题归约（Problem Reduction）**

平凡下界

- 直观：对问题的输入中必须要处理的项及必须要输出的项进行计数

平凡下界

- 直观：对问题的输入中必须要处理的项及必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

平凡下界

- 直观：对问题的输入中必须要处理的项及必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$

平凡下界

- 直观：对问题的输入中必须要处理的项及对必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n+ a_{n-1} x^{n-1} +\ldots+a_0$

$\Omega(n)$: 需要处理所有的系数

平凡下界

- 直观：对问题的输入中必须要处理的项及对必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n+ a_{n-1} x^{n-1} +\ldots+a_0$

$\Omega(n)$: 需要处理所有的系数

3. 计算两个n阶方阵乘积： $\Omega(n^2)$

平凡下界

- 直观：对问题的输入中必须要处理的项及对必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n+ a_{n-1} x^{n-1} +\ldots+a_0$

$\Omega(n)$: 需要处理所有的系数

3. 计算两个n阶方阵乘积： $\Omega(n^2)$

必须处理输入矩阵中的$2n^2$个元素，并且生成乘积中的n^2个元素

平凡下界

- 直观：对问题的输入中必须要处理的项及对必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n+ a_{n-1} x^{n-1} +\ldots+a_0$

$\Omega(n)$: 需要处理所有的系数

3. 计算两个n阶方阵乘积： $\Omega(n^2)$

必须处理输入矩阵中的$2n^2$个元素，并且生成乘积中的n^2个元素

4. n个城市的旅行商问题：

平凡下界

- 直观：对问题的输入中必须要处理的项及对必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$

$\Omega(n)$: 需要处理所有的系数

3. 计算两个n阶方阵乘积： $\Omega(n^2)$

必须处理输入矩阵中的$2n^2$个元素，并且生成乘积中的n^2个元素

4. n个城市的旅行商问题：

输入：$n(n-1)/2$个城市间的距离；输出：最优路线的n个城市列表

平凡下界

- 直观：对问题的输入中必须要处理的项及对必须要输出的项进行计数

例：

1. 生成n个元素的所有排列： $\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$

$\Omega(n)$: 需要处理所有的系数

3. 计算两个n阶方阵乘积： $\Omega(n^2)$

必须处理输入矩阵中的$2n^2$个元素，并且生成乘积中的n^2个元素

4. n个城市的旅行商问题：$\Omega(n^2)$

输入：$n(n-1)/2$个城市间的距离；输出：最优路线的n个城市列表

平凡下界

- 直观：对问题的输入中必须要处理的项及对必须要输出的项进行计数

例：

1. 生成n个元素的所有排列：　$\Omega(n!)$

2. 在给定点x计算 n次多项式 $p(x)=a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$

$\Omega(n)$: 需要处理所有的系数

3. 计算两个n阶方阵乘积：　$\Omega(n^2)$

必须处理输入矩阵中的$2n^2$个元素，并且生成乘积中的n^2个元素

4. n个城市的旅行商问题：$\Omega(n^2)$

平凡下界往往过小，因而用处不大

输入：n(n-1)/2个城市间的距离；输出：最优路线的n个城市列表

下界

- 平凡下界（**Trivial lower bound**）
- 信息论论证（**Information-Theoretic Arguments**）
- 对手论证 （**Adversary Arguments**）
- 问题归约（**Problem Reduction**）

信息论论证

- 根据算法必须处理的信息量来建立下界

信息论论证

- 根据算法必须处理的信息量来建立下界
- 称球问题

信息论论证

- 根据算法必须处理的信息量来建立下界
- 称球问题
  - 已知 3^k 个外表一样的球中有**1个球比其他3^k-1个球重**，用一个天平找出重的那个球

信息论论证

- 根据算法必须处理的信息量来建立下界
- 称球问题
  - 已知 3^k 个外表一样的球中有1个球比其他3^k-1个球重，用一个天平找出重的那个球
  - 需要称 k次

信息论论证

- 根据算法必须处理的信息量来建立下界
- 称球问题
  - 已知 3^k 个外表一样的球中有1个球比其他 3^k-1个球重，用一个天平找出重的那个球
  - 需要称 k次
- 猜数问题
  - 某人在1和n之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数

信息论论证

- 根据算法必须处理的信息量来建立下界
- 称球问题
  - 已知 3^k 个外表一样的球中有1个球比其他3^k-1个球重，用一个天平找出重的那个球
  - 需要称 k次
- 猜数问题
  - 某人在1和n之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数
  - $\lceil \log_2 n \rceil$

信息论论证

- 根据算法必须处理的信息量来建立下界
- 称球问题
  - 已知 3^k 个外表一样的球中有1个球比其他3^k-1个球重，用一个天平找出重的那个球
  - 需要称 k次
- 猜数问题
  - 某人在1和n之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数
  - $\lceil \log_2 n \rceil$
- 决策树（decision tree）

排序问题下界

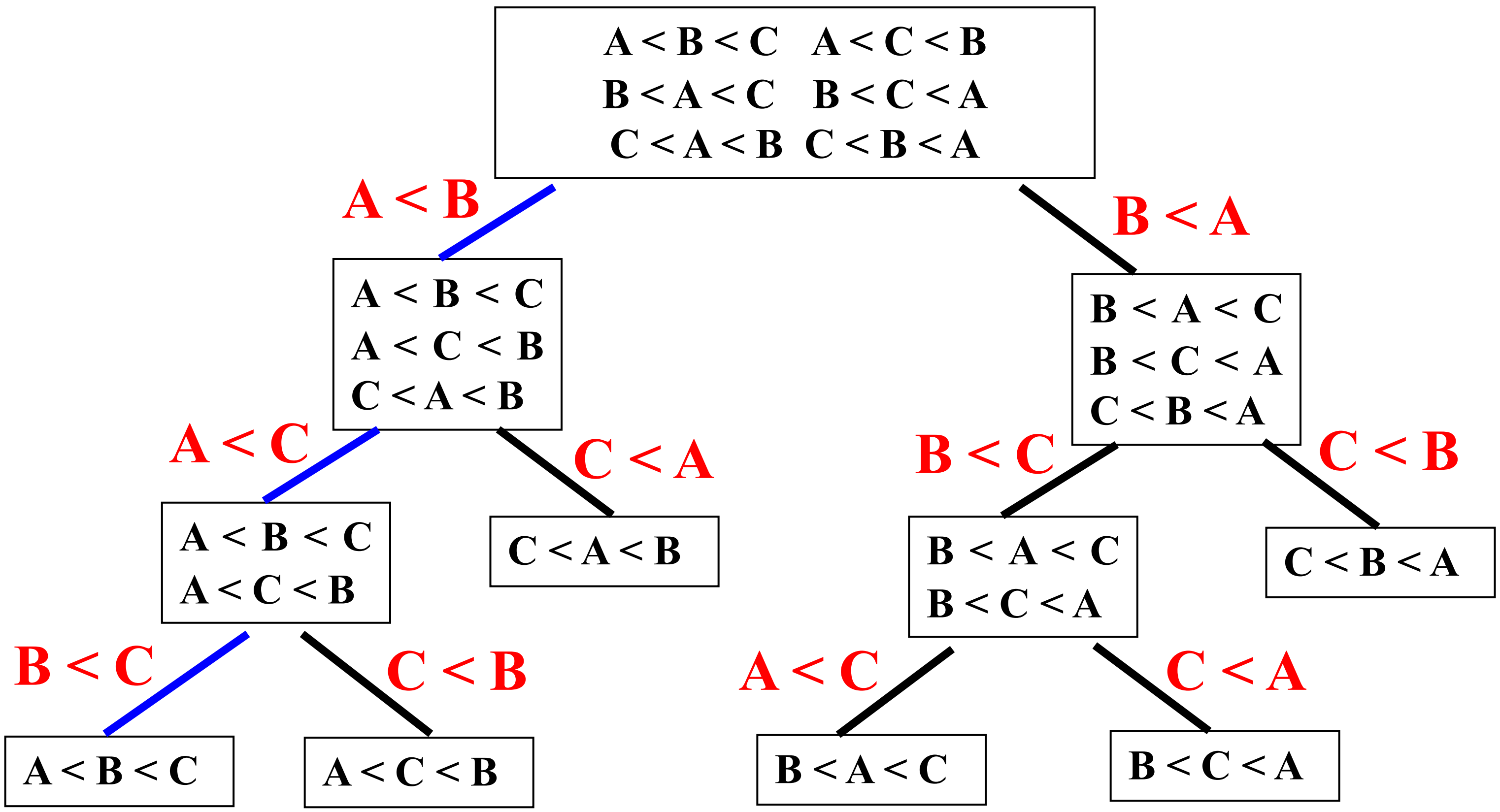
- 基于比较的排序：通过比较来对数列排序
  - 如: **Mergesort, Quicksort**

排序问题下界

- 基于比较的排序：通过比较来对数列排序
  - 如: **Mergesort, Quicksort**
- 决策树：描述排序过程中所需的比较

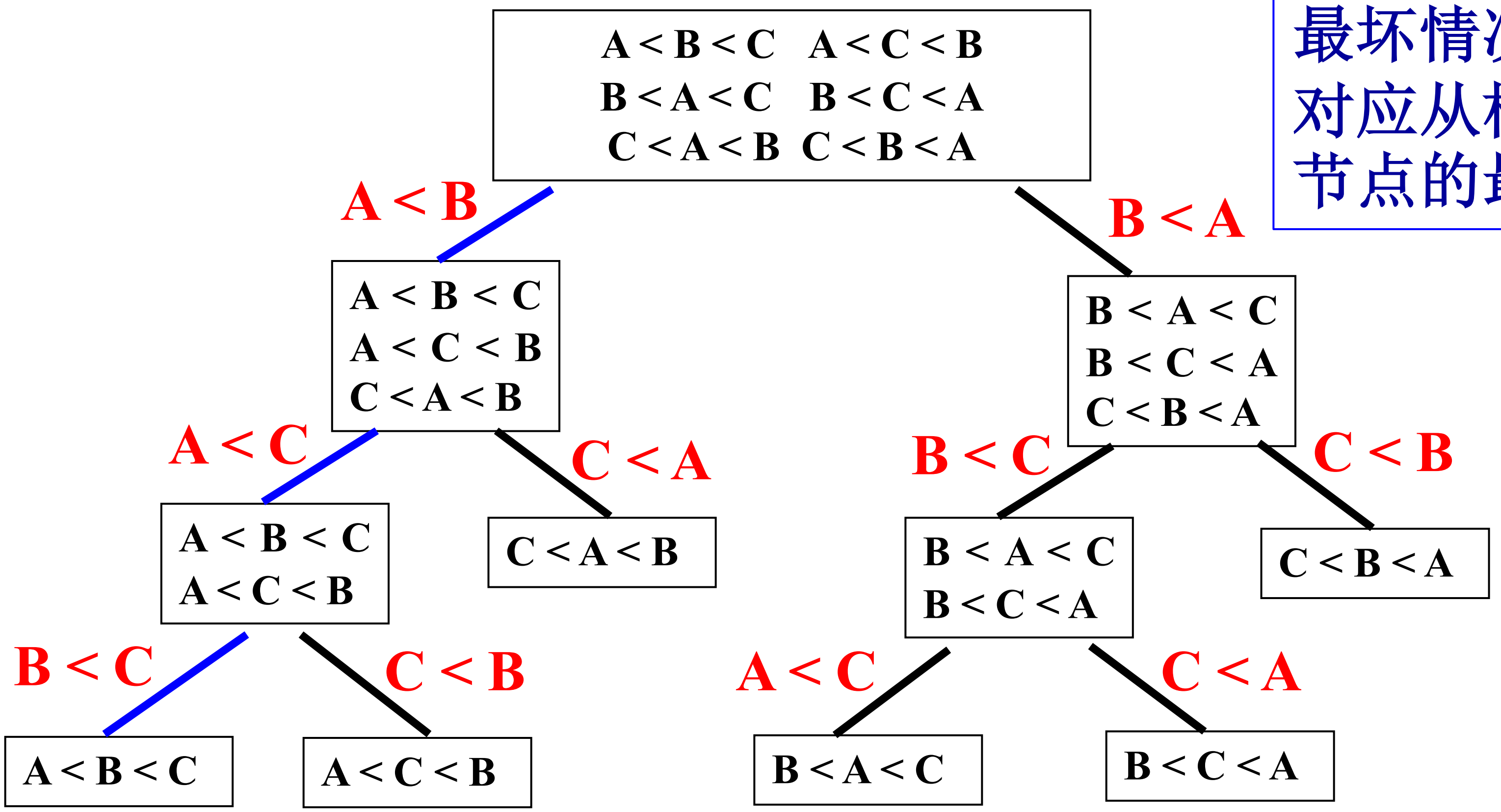
排序问题下界

- 基于比较的排序：通过比较来对数列排序
  - 如: Mergesort, Quicksort
- 决策树：描述排序过程中所需的比较



排序问题下界

- 基于比较的排序：通过比较来对数列排序
  - 如: Mergesort, Quicksort
- 决策树：描述排序过程中所需的比较



最坏情况下的时间对应从根结点到叶节点的最长路径

基于比较的排序下界

定理 1. 任意一个基于比较的排序算法在最坏情况下必须使用 $\Omega(n\log n)$ 次比较对n个元素进行排序。

基于比较的排序下界

定理 1. 任意一个基于比较的排序算法在最坏情况下必须使用 $\Omega(n\log_2 n)$ 次比较对n个元素进行排序。

- 高度为k的二叉树最多有 2^k个叶子节点

$$\log_2(n!) = \log_2(n\,(n-1)\,(n-2)\,\ldots\,2\cdot 1)$$

$$= \log_2 n+\log_2(n-1)+\log_2(n-2)+\ \ldots\ \log_2(2)+\log_2(1)$$

$$\geq \log_2 n+\log_2(n-1)+\log_2(n-2)+\ldots+\log_2(n/2)$$

$$\geq n/2\ \log_2(n/2) = n/2\ (\log n - 1) = \ \Omega(n\log_2 n)$$

下界

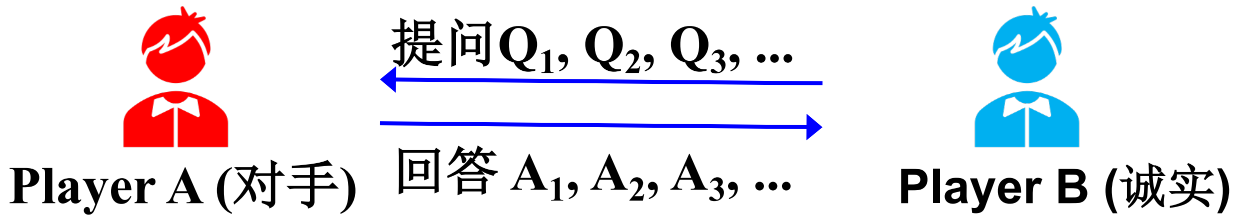
- 平凡下界（**trivial lower bound**）
- 信息论论证（**Information-Theoretic Arguments**）
- 对手论证 （**Adversary Arguments**）
- 问题归约（**Problem Reduction**）

对手论证——猜数问题

- 猜数问题
  - 某人在1和 n 之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数

对手论证——猜数问题

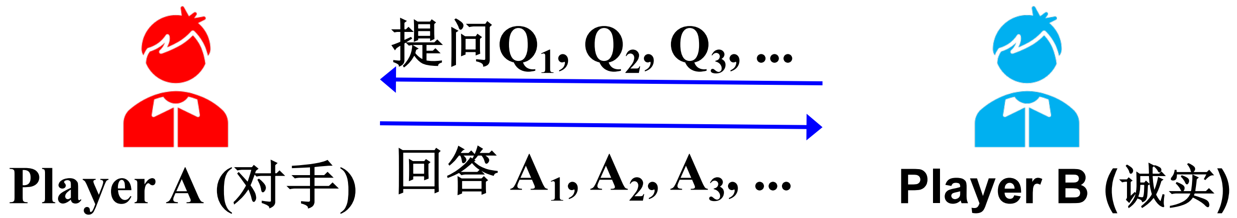
- 猜数问题
  - 某人在1和 n 之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数



从{1,…, n}中选择一个数字 x

对手论证——猜数问题

- 猜数问题
  - 某人在1和 n 之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数

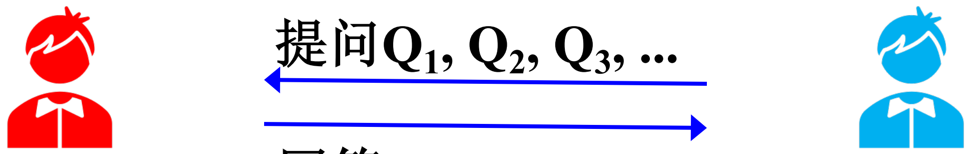


从{1,…, n}中选择一个数字 x

- **Player A**：希望尽可能多提问
  - 尽可能地延长问题序列

对手论证——猜数问题

- 猜数问题
  - 某人在1和 n 之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数



提问 $Q_1, Q_2, Q_3, \ldots$

回答 $A_1, A_2, A_3, \ldots$

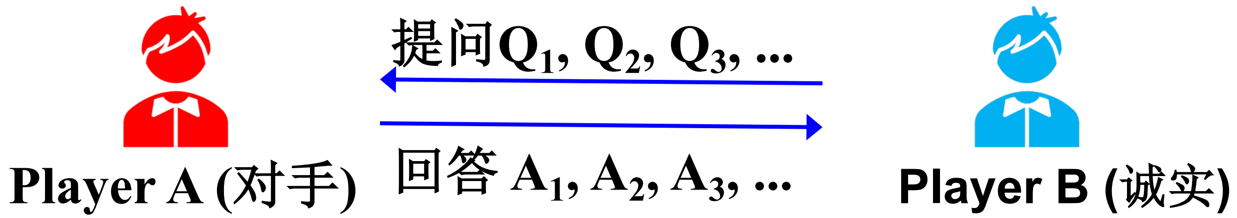
Player A (对手) **Player B (诚实)**

从{1,…, n}中选择一个数字 x

- **Player A**：希望尽可能多提问
  - 尽可能地延长问题序列
  - 给出的回答与前面的回答一致（遵守一种策略）

对手论证——猜数问题

- 猜数问题
  - 某人在1和 n 之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数

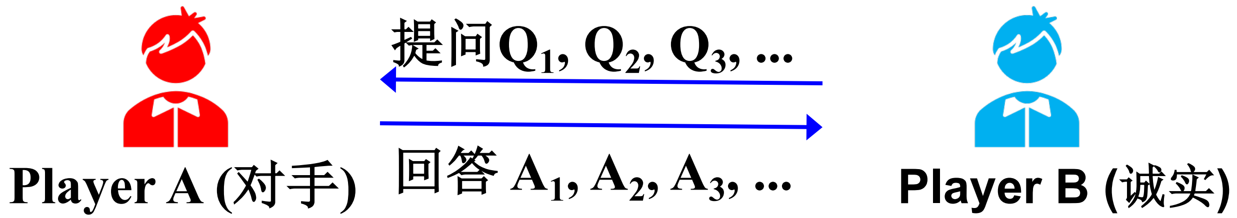


从{1,…, n}中选择一个数字 x

- **Player A**：希望尽可能多提问
  - 尽可能地延长问题序列
  - 给出的回答与前面的回答一致（遵守一种策略）
- 把一个n 元素的削减为一个单元素集合

对手论证——猜数问题

- 猜数问题
  - 某人在1和 n 之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数

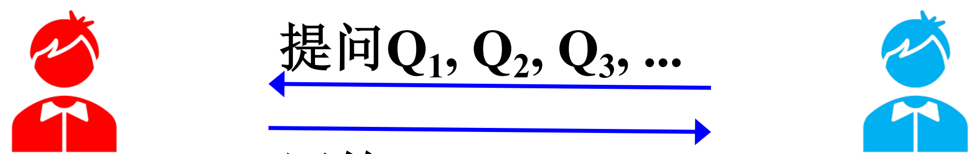


从{1,…, n}中选择一个数字 x

- **Player A**：希望尽可能多提问
  - 尽可能地延长问题序列
  - 给出的回答与前面的回答一致（遵守一种策略）
- 把一个n 元素的削减为一个单元素集合
  - 给出答案把数字最多的集合留下来
  - 留下的集合要和本次答案以及所有前面给出的答案一致

对手论证——猜数问题

- 猜数问题
  - 某人在1和 n 之间选择了一个正整数，通过向他提一些能够回答是或否的问题，推导出这个数



- **Player A**：希望尽可能多提问
  - 尽可能地延长问题序列
  - 给出的回答与前面的回答一致（遵守一种策略）
- 把一个n 元素的削减为一个单元素集合
  - 给出答案把数字最多的集合留下来
  - 留下的集合要和本次答案以及所有前面给出的答案一致

在最坏情况下，任何算法至少要问$\lceil \log_2 n \rceil$个问题

对手论证——猜数问题

令 $S_0=\{1, 2, \ldots, n\}$, S_i 为在 Q_i 回答后所剩的侯选元素集合。

对手论证——猜数问题

令 $S_0 = \{1, 2, \ldots, n\}$, S_i 为在 Q_i 回答后所剩的侯选元素集合。

假设 Q_i 是关于侯选元素 x 的问题，$A_i(x)$ 为 对 Q_i 的回答。

对手论证——猜数问题

令 $S_0=\{1, 2, \ldots, n\}$, S_i 为在 Q_i 回答后所剩的侯选元素集合。

假设 Q_i 是关于侯选元素 x 的问题，$A_i(x)$ 为对 Q_i 的回答。

例：n=100, Q_1=“是否 $x \le 50$?”，则 $A_1(40)$= “是”, $A_1(60)$= “否”。

对手论证——猜数问题

令 $S_0=\{1, 2,\ldots, n\}$, S_i 为在 Q_i 回答后所剩的侯选元素集合。

假设 Q_i 是关于侯选元素 x 的问题，$A_i(x)$ 为对 Q_i 的回答。

例：n=100, Q_1=“是否 $x \le 50$?”，则 $A_1(40)$= “是”, $A_1(60)$= “否”。

定义：
$$Y_i = \{x \in S_{i-1} \mid A_i(x) = \text{yes}\},$$
$$N_i = \{x \in S_{i-1} \mid A_i(x) = \text{no}\}.$$

对手论证——猜数问题

令 $S_0=\{1, 2,\ldots, n\}$, S_i 为在 Q_i 回答后所剩的侯选元素集合。

假设 Q_i 是关于侯选元素 x 的问题，$A_i(x)$ 为对 Q_i 的回答。

例：n=100, Q_1="是否 $x \le 50$?"，则 $A_1(40)$= "是", $A_1(60)$= "否"。

定义：$Y_i = \{x \in S_{i-1} \mid A_i(x) = \text{yes}\},$

$N_i = \{x \in S_{i-1} \mid A_i(x) = \text{no}\}.$

有：$|Y_i| + |N_i| = |S_{i-1}|$

对手论证——猜数问题

令 $S_0=\{1, 2, \ldots, n\}$, S_i 为在 Q_i 回答后所剩的侯选元素集合。

假设 Q_i 是关于侯选元素 x 的问题，$A_i(x)$ 为 对 Q_i 的回答。

例：n=100, Q_1=“是否 $x \le 50$?”，则 $A_1(40)$= “是”, $A_1(60)$= “否”。

定义：$Y_i = \{x \in S_{i-1} \mid A_i(x) = \text{yes}\}$,

$N_i = \{x \in S_{i-1} \mid A_i(x) = \text{no}\}$.

有：$|Y_i| + |N_i| = |S_{i-1}|$

由鸽巢原理知，Y_i与N_i中至少有一个包含至少 $\left\lceil \frac{|S_{i-1}|}{2} \right\rceil$个元素。

对手论证——猜数问题

令 $S_0=\{1, 2,\ldots, n\}$, S_i 为在 Q_i 回答后所剩的侯选元素集合。
假设 Q_i 是关于侯选元素 x 的问题，$A_i(x)$ 为对 Q_i 的回答。

例：n=100, Q_1=“是否 $x \le 50$?”，则 $A_1(40)$= “是”, $A_1(60)$= “否”。

定义：$Y_i = \{x \in S_{i-1} \mid A_i(x) = \text{yes}\},$

$N_i = \{x \in S_{i-1} \mid A_i(x) = \text{no}\}.$

有：$|Y_i| + |N_i| = |S_{i-1}|$

由鸽巢原理知，Y_i与N_i中至少有一个包含至少 $\left\lceil \frac{|S_{i-1}|}{2} \right\rceil$个元素。

Player A 的策略：回答每个问题 Q_i 时，总是让侯选元素 x 落入 Y_i和N_i中的最大一个。

对手论证——猜数问题

- **Player A 的策略**：回答每个问题 Q_i 时，总是让候选元素 x 落入 Y_i 和 N_i 中的最大一个。

对手论证——猜数问题

- **Player A 的策略**：回答每个问题 Q_i 时，总是让候选元素 x 落入Y_i和N_i中的最大一个。

即：若 $|Y_i| \geq |N_i|$，则回答为“是”；否则为“否”。

对手论证——猜数问题

- **Player A 的策略**：回答每个问题 Q_i 时，总是让候选元素 x 落入 Y_i 和 N_i 中的最大一个。

即：若 $|Y_i| \geq |N_i|$，则回答为“是”；否则为“否”。

因此，$S_i = \begin{cases} Y_i & \text{if } |Y_i| \geq |N_i| \\ N_i & \text{if } |Y_i| < |N_i| \end{cases}$，得 $|S_i| \geq \left\lceil \frac{|S_{i-1}|}{2} \right\rceil$，$i \geq 1$

对手论证——猜数问题

- **Player A 的策略**：回答每个问题 Q_i 时，总是<u>让候选元素 x 落入 Y_i 和 N_i 中的最大一个</u>。

即：若 $|Y_i| \geq |N_i|$，则回答为“是”；否则为“否”。

因此，$S_i = \begin{cases} Y_i & \text{if } |Y_i| \geq |N_i| \\ N_i & \text{if } |Y_i| < |N_i| \end{cases}$，得 $|S_i| \geq \left\lceil \frac{|S_{i-1}|}{2} \right\rceil$，$i \geq 1$

令 $b_i = |S_i|$，得 $b_i \geq \lceil b_{i-1}/2 \rceil$，$i \geq 1$

对手论证——猜数问题

- **Player A 的策略**：回答每个问题 Q_i 时，总是让候选元素 x 落入 Y_i 和 N_i 中的最大一个。

即：若 $|Y_i| \geq |N_i|$，则回答为“是”；否则为“否”。

因此，$S_i = \begin{cases} Y_i & \text{if } |Y_i| \geq |N_i| \\ N_i & \text{if } |Y_i| < |N_i| \end{cases}$，得 $|S_i| \geq \left\lceil \frac{|S_{i-1}|}{2} \right\rceil$，$i \geq 1$

令 $b_i = |S_i|$，得 $b_i \geq \lceil b_{i-1}/2 \rceil$，$i \geq 1$

有 $b_0 = n$，$b_1 \geq \lceil b_0/2 \rceil = \lceil n/2 \rceil, b_2 \geq \lceil b_1/2 \rceil \geq \left\lceil \frac{\lceil n/2 \rceil}{2} \right\rceil = \left\lceil \frac{n}{2^2} \right\rceil$

$\ldots, b_i \geq \left\lceil \frac{n}{2^i} \right\rceil$

对手论证——猜数问题

- **Player A 的策略**：回答每个问题 $\mathbf{Q_i}$ 时，总是让候选元素 x 落入$\mathbf{Y}_i$和$\mathbf{N}_i$中的最大一个。

即：若 $|\mathbf{Y}_i| \geq |\mathbf{N}_i|$，则回答为“是”；否则为“否”。

因此，$S_i = \begin{cases} Y_i & \text{if } |Y_i| \geq |N_i| \\ N_i & \text{if } |Y_i| < |N_i| \end{cases}$，得 $|S_i| \geq \left\lceil \frac{|S_{i-1}|}{2} \right\rceil$，$i \geq 1$

令 $b_i = |S_i|$，得 $b_i \geq \lceil b_{i-1}/2 \rceil$，$i \geq 1$

有 $b_0 = n$，$b_1 \geq \lceil b_0/2 \rceil = \lceil n/2 \rceil$, $b_2 \geq \lceil b_1/2 \rceil \geq \left\lceil \frac{\lceil n/2 \rceil}{2} \right\rceil = \left\lceil \frac{n}{2^2} \right\rceil$

$\ldots$, $b_i \geq \left\lceil \frac{n}{2^i} \right\rceil$

整个过程终止于 $\mathbf{b}_i=\mathbf{1}$，此时$\lceil n/2^i \rceil = 1$，得 $i = \lceil \log_2 n \rceil$。

对手论证——猜数问题

- **Player A 的策略**：回答每个问题 Q_i 时，总是让候选元素 x 落入 Y_i和N_i中的最大一个。

即：若 $|Y_i| \geq |N_i|$，则回答为“是”；否则为“否”。

因此， $S_i = \begin{cases} Y_i & \text{if } |Y_i| \geq |N_i| \\ N_i & \text{if } |Y_i| < |N_i| \end{cases}$，得 $|S_i| \geq \left\lceil \frac{|S_{i-1}|}{2} \right\rceil$，$i \geq 1$

令 $b_i = |S_i|$，得 $b_i \geq \lceil b_{i-1}/2 \rceil$，$i \geq 1$

有 $b_0 = n$，$b_1 \geq \lceil b_0/2 \rceil = \lceil n/2 \rceil$, $b_2 \geq \lceil b_1/2 \rceil \geq \left\lceil \frac{\lceil n/2 \rceil}{2} \right\rceil = \left\lceil \frac{n}{2^2} \right\rceil$

$\ldots, b_i \geq \left\lceil \frac{n}{2^i} \right\rceil$

整个过程终止于 $b_i=1$，此时$\lceil n/2^i \rceil = 1$，得 $i = \lceil \log_2 n \rceil$。

在最坏情况下，任何算法至少要问$\lceil \log_2 n \rceil$个问题。

最大值确定问题

定理3. 任意一个基于比较的最大值确定算法均至少需要n-1次比较从n个元素中确定最大值。

证明：假设x是最大值，则x以外的每个元素至少输掉与其他元素的一次比较。
由于每次比较最多产生一个输者。
故至少需要n-1次比较。

- 对最小值确定算法同样成立。

最大值最小值确定问题

- 给定**n**个元素（假设**n**为偶数），计算最大值与最小值

最大值最小值确定问题

- 给定**n**个元素（假设**n**为偶数），计算最大值与最小值
  - 使用 **3n/2−2**次比较可计算最大值与最小值

最大值最小值确定问题

- 给定**n**个元素（假设**n**为偶数），计算最大值与最小值
  - 使用 **3n/2–2**次比较可计算最大值与最小值

算法思想：

输入：**n**个元素 **x[1], x[2], …, x[n]**

输出：**max, min**

最大值最小值确定问题

- 给定n个元素（假设n为偶数），计算最大值与最小值
  - 使用 **3n/2–2**次比较可计算最大值与最小值

算法思想：

输入：**n**个元素 **x[1], x[2], …, x[n]**

输出：**max, min**

1. 进行**n/2**次比较： **x[1]**与**x[2], x[3]**与**x[4], …, x[n-1]**与**x[n]**；
2. 对**n/2**次比较所得的**n/2**个最大值求最大值 **max**; 共**n/2-1**次比较；
3. 对**n/2**次比较所得的**n/2**个最小值求最小值**min**；共**n/2-1**次比较。

最大值最小值确定问题

- 给定n个元素（假设n为偶数），计算最大值与最小值
  - 使用 $3n/2-2$ 次比较可计算最大值与最小值

算法思想：

输入：n个元素 x[1], x[2], …, x[n]

输出：max, min

是否最优算法？

1. 进行n/2次比较： x[1]与x[2], x[3]与x[4], …, x[n-1]与x[n]；
2. 对n/2次比较所得的n/2个最大值求最大值 max; 共n/2-1次比较；
3. 对n/2次比较所得的n/2个最小值求最小值min；共n/2-1次比较。

共需要 $n/2+(n/2-1)+(n/2-1)=3n/2-2$ 次比较

最大值最小值确定问题

- 给定n个元素（假设n为偶数），计算最大值与最小值
  - 使用 3n/2–2次比较可计算最大值与最小值

算法思想：

输入：n个元素 x[1], x[2], …, x[n]

输出：max, min

是最优算法

1. 进行n/2次比较： x[1]与x[2], x[3]与x[4], …, x[n-1]与x[n]；
2. 对n/2次比较所得的n/2个最大值求最大值 max; 共n/2-1次比较；
3. 对n/2次比较所得的n/2个最小值求最小值min；共n/2-1次比较。

共需要n/2+(n/2-1) +(n/2-1)=3n/2-2次比较

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素中确定最大值和最小值至少需要 $(3n/2)–2$ 次比较。

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从**n**个元素中（假设**n**为偶数）确定最大值和最小值至少需要 **$(3n/2)-2$** 次比较。

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素中（假设n为偶数）确定最大值和最小值至少需要 $(3n/2)-2$ 次比较。

- 对手*vs* 算法

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素中（假设n为偶数）确定最大值和最小值至少需要 **$(3n/2)-2$** 次比较。

- 对手*vs* 算法
  - 极大化算法的比较次数

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素中（假设n为偶数）确定最大值和最小值至少需要$(3n/2)-2$ 次比较。

- 对手*vs* 算法
  - 极大化算法的比较次数
  - 极小化获得的信息量

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素中（假设n为偶数）确定最大值和最小值至少需要 $(3n/2)-2$ 次比较。

- 对手*vs* 算法
  - 极大化算法的比较次数
  - 极小化获得的信息量
  - 遵守一定的策略，保持与前面结果的一致性

最大值最小值问题下界

- 除了最大值 **max**，每个元素在比较中至少输一次
- 除了最小值**min**，每个元素在比较中至少赢一次。

最大值最小值问题下界

- 除了最大值 max，每个元素在比较中至少输一次
- 除了最小值min，每个元素在比较中至少赢一次。
- 对每个元素根据以下四种情况进行标记：

N: 从未参与任何比较 ； **W**: 从未输过任何比较

L:从未赢过任何比较；**WL**: 赢了至少一次，也输了至少一次

最大值最小值问题下界

- 除了最大值 max，每个元素在比较中至少输一次
- 除了最小值min，每个元素在比较中至少赢一次。
- 对每个元素根据以下四种情况进行标记：

N: 从未参与任何比较； **W**: 从未输过任何比较

L:从未赢过任何比较； **WL**: 赢了至少一次，也输了至少一次

比较过程中标记会变化

最大值最小值问题下界

- 除了最大值 max，每个元素在比较中至少输一次
- 除了最小值min，每个元素在比较中至少赢一次。
- 对每个元素根据以下四种情况进行标记：

N: 从未参与任何比较； W: 从未输过任何比较

L:从未赢过任何比较；WL: 赢了至少一次，也输了至少一次

- 结论
  - 标记为 W 的元素不可能成为最小值
  - 标记为 L 的元素不可能成为最大值

比较过程中
标记会变化

最大值最小值问题下界

- 除了最大值 max，每个元素在比较中至少输一次
- 除了最小值min，每个元素在比较中至少赢一次。
- 对每个元素根据以下四种情况进行标记：

N: 从未参与任何比较； **W**: 从未输过任何比较

L:从未赢过任何比较；**WL**: 赢了至少一次，也输了至少一次

比较过程中
标记会变化

- 结论
  - 标记为 **W** 的元素不可能成为最小值
  - 标记为 **L** 的元素不可能成为最大值
  - 标记为 **WL**的元素不会改变标记

对手的策略

compare labeled (x,y)	comparison results	changing labels to	obtain this many units of information
(N,N)	x > y	(W,L)	2
(W,N) or (WL,N)	x > y	(W,L) or (WL,L)	1
(L,N)	x < y	(L,W)	1
(W,W)	x > y	(W,WL)	1
(L,L)	x > y	(WL,L)	1
(W,L) or (WL,L) or (W,WL)	x > y	no change	0
(WL,WL)		no change	0

最大信息量为2

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素（假设n为偶数）中确定最大值和最小值至少需要 **$(3n/2)-2$** 次比较。

证明：算法初始把所有元素标记为N，

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素（假设n为偶数）中确定最大值和最小值至少需要 $(3n/2)-2$ 次比较。

证明：算法初始把所有元素标记为N，且算法运行过程中必须标记n-1次W和n-1次L，因此至少需要 $2n-2$ "信息单位"。

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素（假设n为偶数）中确定最大值和最小值至少需要 $(3n/2)-2$ 次比较。

证明：算法初始把所有元素标记为N，且算法运行过程中必须标记n-1次W和n-1次L，因此至少需要 $2n-2$ “信息单位”。
每次比较中能得到的最大信息量为2，且只当比较两个标记为N的元素时产生。但此种情况最多产生 n/2次，共n个信息单位 。

最大值最小值问题下界

定理4. 在最坏情况下，任意一个基于比较的算法从n个元素（假设n为偶数）中确定最大值和最小值至少需要 **$(3n/2)-2$** 次比较。

证明：算法初始把所有元素标记为N，且算法运行过程中必须标记n-1次W和n-1次L，因此至少需要$2n-2$ “信息单位”。

每次比较中能得到的最大信息量为2，且只当比较两个标记为N的元素时产生。但此种情况最多产生 n/2次，共n个信息单位 。

除此以外的其他比较最多产生1个信息量，因此对于剩下的n-2个信息量，最坏情况下需要 n-2次比较。

因此，最坏情况下，至少需要 $n/2 + n-2 = 3n/2 - 2$次比较。

计算第二大值问题

- **给定n个元素（假设n为偶数），计算第二大值**

计算第二大值问题

- 给定n个元素（假设n为偶数），计算第二大值
- **Naive algorithm：**

1. 找到最大值（**n-1**次比较）；
2. 去掉最大值，在剩下元素找到最大值（**n-2**次比较）。

共需要 **2n-3**次比较。

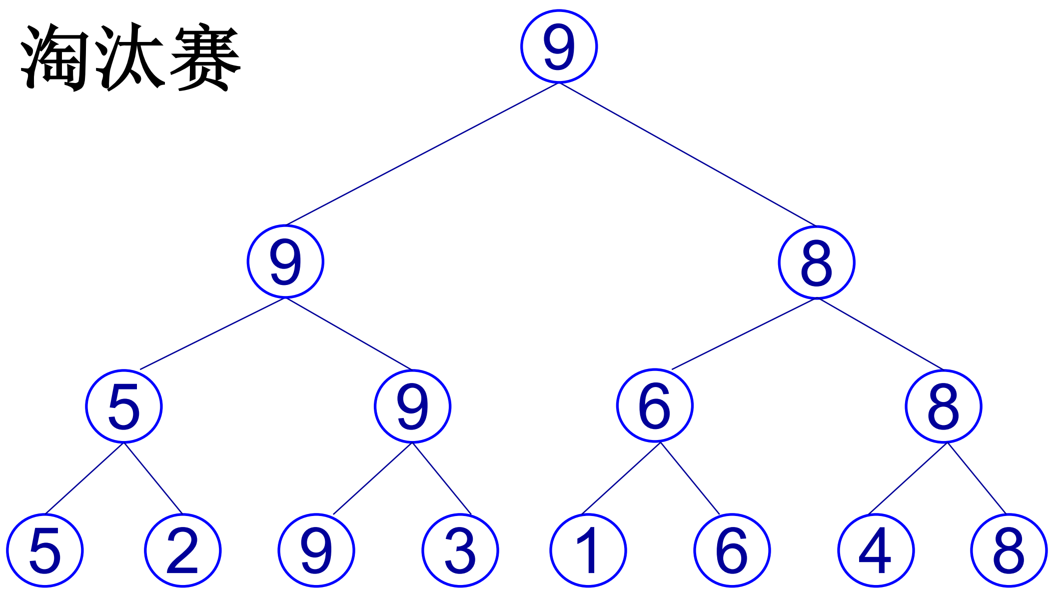
计算第二大值问题

- 给定n个元素（假设n为偶数），计算第二大值
- **Naive algorithm：**

1. 找到最大值（**n-1**次比较）；
2. 去掉最大值，在剩下元素找到最大值（**n-2**次比较）。

共需要 **2n-3**次比较。

- 淘汰赛



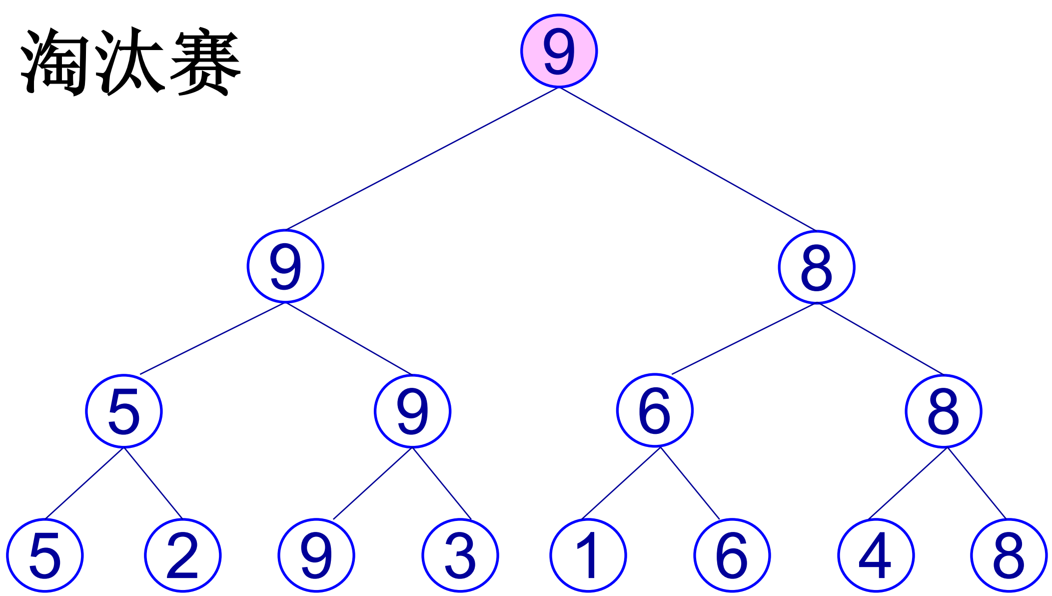
计算第二大值问题

- 给定n个元素（假设n为偶数），计算第二大值
- **Naive algorithm:**

1. 找到最大值（**n-1**次比较）；
2. 去掉最大值，在剩下元素找到最大值（**n-2**次比较）。

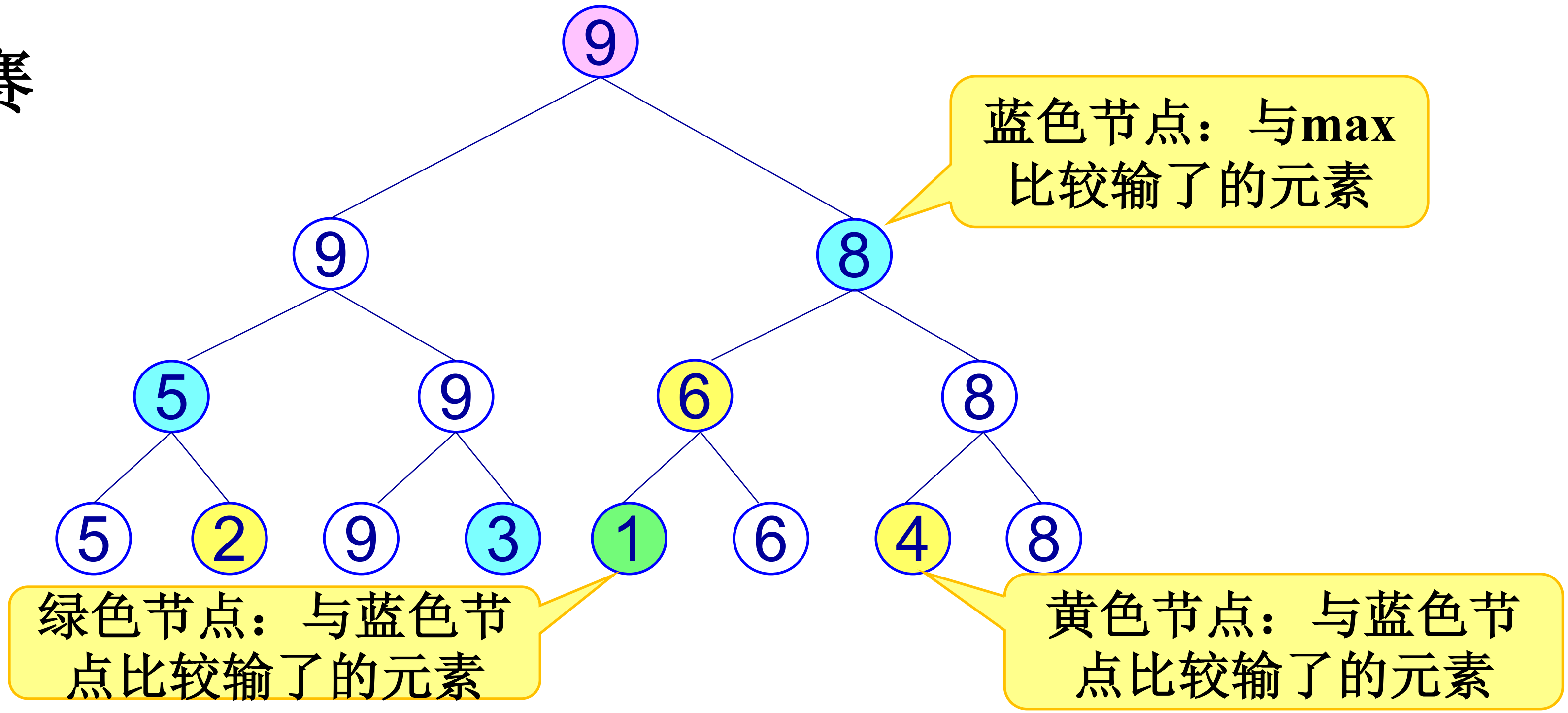
共需要 **2n-3**次比较。

- 淘汰赛



计算第二大值问题

- 淘汰赛



输入：**n**个元素 **x[1], x[2], …, x[n]** （假设$n=2^k$, $k \geq 1$）

输出：第二大值

1. 利用淘汰赛求得最大值**max**（**n-1**次比较）；
2. 从所有与**max**进行比较输了的元素中找到最大值（$\log_2 n-1$次比较）

共需要$n-1+\log_2 n-1 = n+\log_2 n - 2$次比较。

计算第二大值问题下界

定理 5. 在最坏情况下，任意一个基于比较的算法从n个元素（假设n为偶数）中确定第二大值至少需要 $n-2+\log_2 n$ 次比较。

总结

- 平凡下界（**Trivial lower bound**）
- 信息论论证（**Information-Theoretic Arguments**）
  - 决策树
- 对手论证 （**Adversary Arguments**）
  - 最大值最小值问题
  - 第二大值问题