# Selection

## The Selection Problem

- 问题定义
  - 设$E$是一个包含$n$个，key来自于线性有序集合的，元素的数组，设$k$为一个正整数使$1 \leq k \leq n$. 选择问题就是找到$E$中第$k$小的元素.
- 特殊情况
  - 找最大值/最小值--$k = n$ or $k = 1$
  - 找中位数--$k = \frac{n}{2}$

## 找出最大元素的时间复杂度下界

对任何只能比较和copy的算法，在最坏情况下，要找到一个有$n$项的数组的最大项至少要比较$n - 1$次

- 证明：假设有一个有$n$个不同项的数组。只能通过一次发现一项比另一个项小才能排除这个项为最大项的可能，所以$n - 1$项都得是在算法中被"小看"过才行。而每次比较只能知道一项较小，所以至少需要$n - 1$次比较。

### 决策树与找最大元素算法下界

因为最大元素所有可能情况就是元素个数$n$，如果假设决策树只有这些叶节点，那么搞出来下界是$\lceil \log n \rceil$，但实际上这个下界不好，实际上决策树有$2^{n-1}$个叶节点，此时我们需要*Adversary Argument*这种新方法来更好地分析下界。

## 对手论证 - Adversary Argument

### Finding *max* and *min*

考虑找最大元素和最小元素问题

- The Strategy
  - 2个元素一组比较，共做$n/2$次比较（如果$n$为奇数就先不比E[n]，即最后一个元素）
  - 在上一步比完的较大元素集里面进行*findMax*操作，较小元素集里面进行*findMin*操作（如果$n$为奇数，那么余下的那个元素同时放入两个集合中）
- Number of Comparisons
  - 偶数$n$：$n/2 + 2(n/2 - 1) = 3n/2 - 2$
  - 奇数$n$：$(n-1)/2 + 2((n-1)/2 + 1 - 1) = \lceil 3n/2 \rceil - 2$
- 证明--Adversary Argument

> *Introduction to Algorithms p.215*
>
> We compare pairs of elements from the input first with each other, and then we compare the smaller with the current minimum and the larger to the current maximum, at a cost of 3 comparisons for every 2 elements.
>
> How we set up initial values for the current minimum and maximum depends on whether n is odd or even. If $n$ is odd, we set both the minimum and maximum to the value of the first element, and then we process the rest of the elements in pairs. If $n$ is even, we perform 1 comparison on the first 2 elements to determine the initial values of the minimum and maximum, and then process the rest of the elements in pairs as in the case for odd $n$.
>
> Let us analyze the total number of comparisons. If $n$ is odd, then we perform $3\lfloor n/2 \rfloor$ comparisons. If $n$ is even, we perform 1 initial comparison followed by $3(n-2)/2$ comparisons, for a total of $3n/2-2$. Thus, in either case, the total number of comparisons is at most $3\lfloor n/2 \rfloor$.

### Unit of Information (一个足够抽象的数学工具)

> *From Wikipedia, the free encyclopedia*
>
> **Units of Information**
>
> In information theory, units of information are also used to measure the entropy of random variables and information contained in messages.
>
> **Information theory**
>
> **Algorithmic Information Theory**
>
> ……

- Max and Min
  - $x$只有当所有非$x$的元素都在某次比较中被"小看"才能成为最大元素
  - $y$只有当所有非$y$的元素都在某次比较中被"高看"才能成为最小元素
- 每次win或者loss被算作one unit of information
  - 任何算法都至少需要$2n-2$ units of information才能确定max和min元素

### Adversary Strategy

| 算法中 $x$ 和 $y$ 的状态 | Adversary reponse | New Status | 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 |
| $WL$ or $WL, L$ or $W, WL$ | $x > y$ | No change | 0 |
| $WL, WL$ | 维持现状 | No change | 0 |

注：$N$代表无状态，$W$代表赢过，$L$代表输过

**The principle**: let the key win if it never lose, or, let the key lose if it never win, and change one value if necessary

### Lower Bound by the Adversary Argument

- 构造要求算法做尽可能多次比较的输入
  - AS要做的：使得每次比较获得的units of information尽量少
    - 只有在当前状态是$N, N$时才能获得2 units of information（对算法设计者最有利的情况）
    - 在其他状态下总是有办法给出Adversary Response使得最多只能获得1 unit of information，在不前后矛盾的前提下
- 从而下界就是$n/2 + n - 2$（偶数情况），units of information总数等于 $n/2 \times 2 + (n-2) \times 1 = 2n - 2$

## Find the 2nd Largest Key

算法--**胜者树 - 锦标赛排序**

画出树即可理解

最长路径的长度是$\lceil \log n \rceil$，其值也是和*max*元素比较过的元素至多的数量。

### 时间复杂度分析

- 任何一个找第二大元素的算法必定先找到了第一大，而找第一大的比较次数是 $n - 1$
- 第二大元素必定在那些直接输给第一大元素的元素里
- 在杀出一条血路，取得冠军的过程中，第一大最多击败了$\lceil \log n \rceil$个元素
- 所以输掉的最多$\lceil \log n \rceil - 1$个，在其中可以找到第二大元素
- 从而总cost是$n + \lceil \log n \rceil - 2$

### 用对手论证进行下届分析

**Theorem**

- 通过比较在n个元素的集合里找第二大元素在最坏情况下最少要比较$n+\lceil\log n\rceil-2$次

**Proof**

- 有一种**对手策略**可以使任何算法需要先比较$\lceil\log n\rceil$来找出*max*元素

#### Weighted Key - 给key加权来构造AS

- $w(x)$表示每个元素的权值，一开始均设为1
- 对手策略

| Case | Adversary reply | Updating of weights |
|---|---|---|
| $w(x)>w(y)$ | $x>y$ | $w(x):=w(x)+w(y);\ w(y):=0$ |
| $w(x)=w(y)>0$ | $x>y$ | $x(x):=w(x)+w(y);\ w(y):=0$ |
| $w(x)<w(y)$ | $x<y$ | $w(y):=w(x)+w(y);\ w(x):=0$ |
| $w(x)=w(y)=0$ | Consistent with previous replies | No change |

#### 证明细节

- 所有元素权值和为$n$
- 假设$x$是最大元素，那么$x$最后应当是唯一的权值非零的元素（唯一性，考虑反证法，假设有个元素weight非零，则要么它没进行过比较，则还不能确认$x$是最大元素，要么和$x$比较获得了其权值，显然不可能），同时，这意味着它获得了所有元素的权值（`Updating of weights`那一栏可见，每一次权值更新对被更新元素$p,q$，总有$w(p)+w(q)$不变，所以$w(x)=n$）
- 通过对手策略，有

$$w_k(x)\le 2w_{k-1}(x)$$

- 设$K$为$x$在与先前未曾lose的元素比较的次数则有（连续使用以上不等式）：

$$n=w_k(x)\le 2^K w_0(x)=2^K$$

- 因而，$K\ge\lceil\log n\rceil$

## 线性时间的选择算法设计

### 找中位数

#### Strategy

- Observation
  - 如果把key数组分成S1, S2两个集合，S1中的元素都小于S2中元素，那么中位数必定落在有更多元素的那个集合里
- Divide-and-Conquer
  - 只有一个集合需要递归处理

#### 调整rank

原始的中位数在分裂过的集合里rank会变，所以要主动调整

#### Partitioning

同quick-sort，比pivot小的放一边，大的放另一边

#### Selection: the Algorithm

- input
- output
- procedure
- element select

关键在于

1. 为了在最坏时间复杂度情况下保证$O(n)$，如何进行partition（见下文）
2. **if** $(|S| \leq 5)$ **return** direct solution，即base case做法，换言之将所有元素分成5个一组，共$\lceil \frac{n}{5} \rceil$组，对于每组的5个元素找出其中位数，显然这是一个常数时间可以解决的问题。

#### 改进Partition

> 对于每一组，将比中位数小的元素画在中位数上方，比中位数大的元素画在中位数的下方。对于每一组的中位数，递归地使用最坏情况线性时间复杂度选择算法求出其中位数。记为$m^*$，并用$m^*$来划分所有中位数。

这样就分出了4块，ABCD

再用$m^*$进行标准的selection的partition，就可以获得大小两部分，再通过参数$k$和两部分的大小关系，找到子问题递归进行。

这种划分方式用一定的额外代价保证了划分的**平衡性**。

#### Divide and Conquer

$$
\begin{aligned}
&\text{if } (k = |S_1| + 1) \\
&\qquad \text{return } m^*; \\
&\text{else if } (k \leq |S_1|) \\
&\qquad \text{select}(S_1, k);\ //\text{recursion} \\
&\text{else} \\
&\qquad \text{return select}(S_2, k - |S_1| - 1);\ //\text{recursion}
\end{aligned}
$$

#### Analysis

略，见课本或CLRS