Selection

Warm up

> **定义 3.1** 选择问题
>
> - 输入：n 个各不相同的元素组成的全序集 $< a_1, a_2, \ldots, a_n >$，参数 $k, 1 \leqslant k \leqslant n$
> - 输出：阶为 k 的元素

Lower bound of Finding Max

即 $k = n$ 的情况，遍历即可

任意基于比较的算法，在 n 个元素中寻找最大元素时，进行的比较次数不会少于 $n - 1$ 次

Proof：一个数不是 Max $\iff$ 在至少一次比较中它是 loser，而找出 Max 即为排除 $n - 1$ 个 loser，但一次比较最多得出一个 loser，故至少需要 $n - 1$ 次比较

Finding max and min

strategy：

- 将元素配对，两两比较（元素个数为奇数个时最后一个元素不比较）
- 比较后的元素分为 larger 和 smaller 两个集合，在 larger 中 Find Max，在 smaller 中 Find Min（元素个数为奇数个时最后一个元素加入 larger 和 smaller）

比较次数：

- 偶数：$\frac{n}{2} + 2\left(\frac{n}{2} - 1\right) = \frac{3n}{2} - 2$
- 奇数：$\frac{n-1}{2} + 2\left(\frac{n-1}{2} + 1 - 1\right) = \left\lceil \frac{3n}{2} \right\rceil - 2$

如何得到下界？Adversary argument

Adversary argument

基于决策树可以得出 Find Max 的一个下界 $\lceil \log n \rceil$，但这个下界太过宽松，需要更精确的下界

引入 Adversary argument 的分析方法

Adversary argument

> 对手论证
>
> 对手论证，一般用于给出问题的下界。
>
> 若用 P 表示所讨论的问题，I 表示问题的输入，A 表示解决问题的基于比较运算的算法，$T(A,I)$ 表示对于输入 I，算法 A 的计算时间复杂性，那么函数
>
> $$U(n)=\min\{\max\{T(A,I)\},\text{for each } I\},\ \text{for each } A\}$$
>
> 表示问题 P 在输入大小为 n 时在最坏情况下的时间下界，它是问题所固有的。
>
> 对手论证的基本思想是对每一个 A 构造一个输入特殊的输入 I'，使 $T(A,I')$ 尽量地大，然后在所有 A 的集合上，求 $T(A,I')$ 的尽量小的下界作为 $f(n)$。

Adversary argument 思想：构造一个特殊的合法输入使得算法的代价尽可能大，最终得到算法代价的下界

Adversary argument 关键：面向问题，即对于解决问题 P 的一切算法 A，有一套通用的策略去构造输入使得 A 的代价尽可能大

Lower bound of Finding max and min

> Unit of information
>
> In information theory, units of information are also used to measure the entropy of random variables and information contained in messages.

对于 Find max and min 问题，unit of information 为

- 某元素在某次比较中输给其他元素
- 某元素在某次比较中胜过其他元素

x 若为 max $\iff$ x 以外的元素均在某次比较中输给别的元素

x 若为 min $\iff$ x 以外的元素均在某次比较中胜过别的元素

则任意算法必须至少获得 $2n-2$ 个 unit of information 才能确定 max 和 min

Adversary strategy：使得算法每一次比较获得尽可能少的 unit of information

对于 Find max and min (N 为位置，W 为胜出，L 为失败)

Status of keys	Adversary response	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
W, L or WL, L, or W, WL	x > y	no change	0
WL, WL	consistent with assigned values	no change	0

策略：让胜利过的元素一直胜利，让失败过的元素一直失败

仅当 N, N 的情况可以通过一次比较得到 2 个 unit of information，这样的比较最多可以进行 $\left\lfloor \frac{n}{2} \right\rfloor$ 次，其余情况均可给出 adversary response 使得每次比较最多获得 1 个 unit of information，故下界是

$$\frac{n}{2} + n - 2 = \frac{3n}{2} - 2 \quad (\text{for even } n)$$

Lower bound of Finding the 2^{nd} Largest Key

BF 解法：进行两次 Find Max，比较次数为 $2n - 3$

优化思路：败给 Max 以外元素的不可能是 2^{nd} Largest，建立胜者树，则在 Max 的上升路径上的败者均是 2^{nd} Largest 的候选。败者最多为 $\lceil \log n \rceil$ 个，即上升路径长度的最大值。

比较次数为 $(n-1) + (\lceil \log n \rceil - 1) = n + \lceil \log n \rceil - 2$

可以使用 adversary argument 证明算法下界为 $n + \lceil \log n \rceil - 2$

为每个 key 定义一个 weight 函数，初始值为 1

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	w(x) := w(x) + w(y); w(y) = 0
w(y) > w(x)	y > x	w(y) := w(x) + w(y); w(x) = 0
w(x) = w(y) = 0	consistent with previous replies	no change

Adversary strategy：保证每次比较，weight 的增长不超过一倍

- weight 的和总为 n
- 令 x 为 max，则结束时 $w(x) = n$
- $w_k(x) \leqslant 2w_{k-1}(x)$
- 令 K 为 x 打败的之前未曾被击败的元素数量，$n = w_K(x) \leqslant 2^K w_0(x) = 2^K$
- $K \geqslant \lceil \log n \rceil$

即 Max 胜利路径上的败者数下界为 $\lceil \log n \rceil$

具体实现：建立大小为 $2n-1$ 的堆结构，在最后 n 个位置放入输入元素，根据堆偏序特性即可追踪 Max 的败者

Finding the Median

Strategy：D&C

Find median 可等价为选择阶为 k 的元素的特殊情况（$k = \frac{n}{2}$）

选择阶为 k 的元素 x 对应一种偏序关系：有 $k-1$ 个元素比 x 小，有 $n-k$ 个元素比 x 大

于是可以将输入基于某个 pivot 划分为左半部分，pivot，右半部分，进行递归的处理

- 若左半部分元素个数为 $k-1$，返回 pivot
- 若左半部分元素个数大于 $k-1$，在左半部分递归地寻找阶为 k 的元素
- 若左半部分元素个数小于 $k-1$，在右半部分递归地寻找阶为 $k-n_1-1$ 的元素（n_1 为左半部分元素个数）

Algorithm

```
Element select(Set S, int k)
    if |S| <= 5
        return direct solution
    else
        Constructing the subset S_1, S_2        // Key issue
        Processing one of S_1, S_2
```

Key issue: 划分的方法

Expected linear time: 类似快排，选择 pivot 后划分为两部分

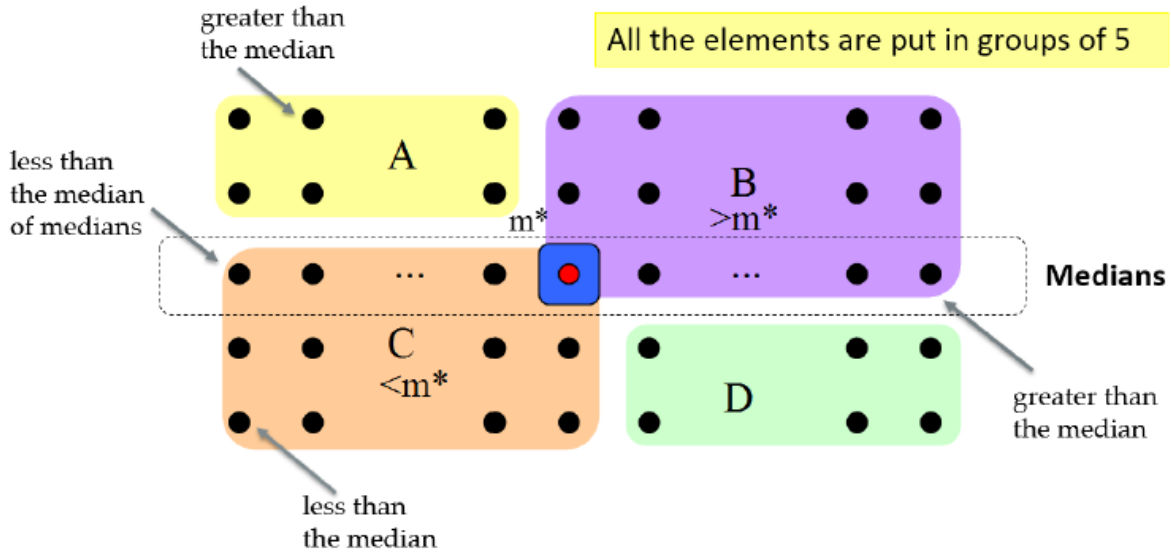
易得在最坏情况下，每次划分中其中一部分为空，算法的最坏情况时间复杂度为 $O(n^2)$

而平均情况下可证得算法平均情况时间复杂度为 $O(n)$ （证明见课本 8.1.2 节）

Worst-case linear time

关键：使得每次划分即使在最坏情况下也不会太不平均

划分思路：将所有元素按 5 个一组划分，共 $\left\lceil \frac{n}{5} \right\rceil$ 组，对于每组，找出其中的中位数，并且将 5 个元素划分为 2 个大于中位数，2 个小于中位数。如下图



递归找出所有中位数的中位数，记为 m^* ，将 m^* 作为 pivot 划分元素，即

$$\text{Let } S_1 = C \cup \{x | x \in A \cup D \text{ and } x < m^*\}$$
$$\text{Let } S_2 = B \cup \{x | x \in A \cup D \text{ and } x > m^*\}$$

之后的处理思路同上

- 如果 $k = |S_1| + 1$ ，返回 m^*

- 如果 $k \leqslant |S_1|$，返回 $\text{select}(S_1, k)$
- 如果 $k > |S_1| + 1$，返回 $\text{select}(S_2, k - |S_1| - 1)$

Analysis

分析为何在最坏情况下仍是线性时间

首先假设 $n = 5(2r + 1)$

则

$$W(n) \leqslant 6\left(\frac{n}{5}\right) + W\left(\frac{n}{5}\right) + 4r + W(7r + 2)$$

- $6\left(\frac{n}{5}\right)$ 为在 5 个数中寻找中位数所需要的比较次数

  > 5 个元素中寻找中位数最多只用比较 6 次
  >
  > 设五个元素为 a, b, c, d, e
  >
  > 首先将 a, b, c 排序，至多需要 3 次排序，设排序后为 $s < m < l$
  >
  > 比较 d, e，不失一般性，设 $d > e$，比较 d, m
  >
  > - 若 $d > m$，e, m 中较大者为中位数
  > - 若 $d < m$，d, s 中较大者为中位数

- $W\left(\frac{n}{5}\right)$ 为递归寻找 m^* 的代价
- $4r$ 为比较 m^* 与 $A \cup D$ 中元素的代价
- $W(7r + 2)$ 为最坏情况，即 $A \cup D$ 中所有元素均划分在同一子集

$$\text{Note: } r \text{ is about } n/10, \text{ and } 0.7n + 2 \text{ is about } 0.7n, \text{ so}$$
$$W(n) \leqslant 1.6n + W(0.2n) + W(0.7n)$$

根据递归树易得 row sums 为 $1.6n, 1.6(0.9)n, 1.6(0.81)n, 1.6(0.729)n, \ldots$，为递减的几何级数，故

$$W(n) = \Theta(n)$$

Lower bound of Finding median

Relation to median

显然任意选择算法必须知道其余所有元素和 median 的关系（否则可构造出反例）

故可定义寻找 median 中的 **crucial comparison**，即建立起某元素与 median 的大小关系的比较

> - Crucial comparison for x: the first comparison where $x > y$, for some $y \geqslant median$, or $x < y$ for some $y \leqslant median$
> - Non-crucial comparison: the comparison between x and y where $x > median$ and $y < median$， or vise versa

Adversary strategy

定义一个 key 为

- L：其值大于 median
- S：其值小于 median
- N：未参加比较

则

Comparands	Adversary's action
N,N	one L, the other S
L,N or N,L	change N to S
S, N or N, S	change N to L

其余情况保持不变

则至少可以分配 $\frac{n-1}{2}$ 个 L 或 S，使得进行的比较均为 Non-crucial comparison。在此之后，若已经有 $\frac{n-1}{2}$ 个 L，之后为 key 分配的值必须小于 median，反之同理，最后一个被分配的值是 median

Lower bound

基于 adversary strategy 可以确保算法至少要做 $\frac{n-1}{2}$ 次 Non-crucial comparison，且至少需要 $n-1$ 次 crucial comparison 以确定其他元素和 median 的关系，故任何寻找中位数的算法至少需要进行 $\frac{3n}{2}-\frac{3}{2}$ 次比较（n 为奇数）