

众数

12. 排序

选取：众数

邓俊辉

deng@tsinghua.edu.cn

选取与中位数

- ❖ **k-selection** 在任意一组可比较大小的元素中，如何由小到大，找到次序为k者？
亦即，在这组元素的非降排序序列s中，找出 $s[k]$

// Excel: large(range, rank)

- ❖ **median** 长度为n的有序序列s中，元素 $s[\lceil \frac{n}{2} \rceil]$ 称作中位数 // 数值上可能有重复
在任意一组可比较大小的元素中，如何找到中位数？

// Excel: median(range)



- ❖ 中位数是k-选取的一个特例；稍后将看到，也是其中难度最大者

众数

❖ **majority** 无序向量中，若有**一半以上**元素同为**m**，则称之为众数

在{ 3, 5, 2, 3, 3 }中，众数为3；然而

在{ 3, 5, 2, 3, 3, 0 }中，却**无**众数

❖ 平凡算法 排序 + 扫描

但进一步地 若限制时间不超过**O(n)**，附加空间不超过**O(1)**呢？

❖ 必要性 众数若存在，则亦必**中位数**

❖ 事实上 只要能够**找出**中位数，即不难**验证**它是否众数

```
template <typename T> bool majority( Vector<T> A, T & maj )  
{ return majEleCheck( A, maj = median( A ) ); }
```

必要条件

- ❖ 然而 在高效的中位数算法未知之前，如何确定众数的候选呢？
- ❖ mode 众数若存在，则亦必频繁数 //Excel: mode(range)

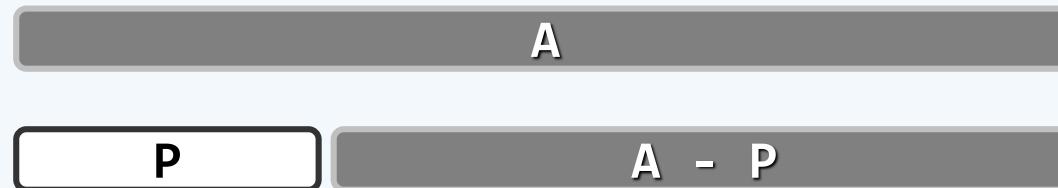
```
template <typename T> bool majority( Vector<T> A, T & maj )  
{ return majEleCheck( A, maj = mode( A ) ); }
```
- ❖ 同样地 mode() 算法难以兼顾时间、空间的高效
- ❖ 可行思路 借助更弱但计算成本更低的必要条件，选出唯一的候选者

```
template <typename T> bool majority( Vector<T> A, T & maj )  
{ return majEleCheck( A, maj = majEleCandidate( A ) ); }
```

减而治之

❖ 若在向量A的前缀P (|P|为偶数) 中，元素x出现的次数恰占半数，则

A有众数仅当，对应的后缀A - P有众数m，且m就是A的众数



❖ 既然最终总要花费 $\Theta(n)$ 时间做验证，故而只需考虑A的确含有众数的两种情况：

1. 若 $x = m$ ，则在排除前缀P之后，m与其它元素在数量上的差距保持不变

(从浓度 50% 的盐水中渗析出 50% 的一部分，剩余部分的浓度仍为 50%)

2. 若 $x \neq m$ ，则在排除前缀P之后，m与其它元素在数量上的差距不致缩小

算法

```
❖ template <typename T> T majEleCandidate( Vector<T> A ) {  
    T maj; //众数候选者  
  
    // 线性扫描：借助计数器c，记录maj与其它元素的数量差额  
  
    for ( int c = 0, i = 0; i < A.size(); i++ )  
        if ( 0 == c ) { //每当c归零，都意味着此时的前缀P可以剪除  
            maj = A[i]; c = 1; //众数候选者改为新的当前元素  
        } else //否则  
            maj == A[i] ? c++ : c--; //相应地更新差额计数器  
  
    return maj; //至此，原向量的众数若存在，则只能是maj——尽管反之不然  
}
```

12. 排序

选取：中位数

中也者，天下之大本也
和也者，天下之达道也

邓俊辉

deng@tsinghua.edu.cn

归并向量的中位数

❖ 任给已经排序的有序向量 S_1 和 S_2

如何快速找出有序向量 $S = S_1 \cup S_2$ 的中位数?

❖ 蛮力: 归并 S_1 和 S_2 , 得到有序向量 S

取出 $S[(|S_1| + |S_2|) / 2]$, 即是 S 的中位数

❖ 如此, 共需 $\mathcal{O}(|S_1| + |S_2|)$ 时间

❖ 这一效率虽不算低, 但毕竟未能充分利用 S_1 和 S_2 的有序性

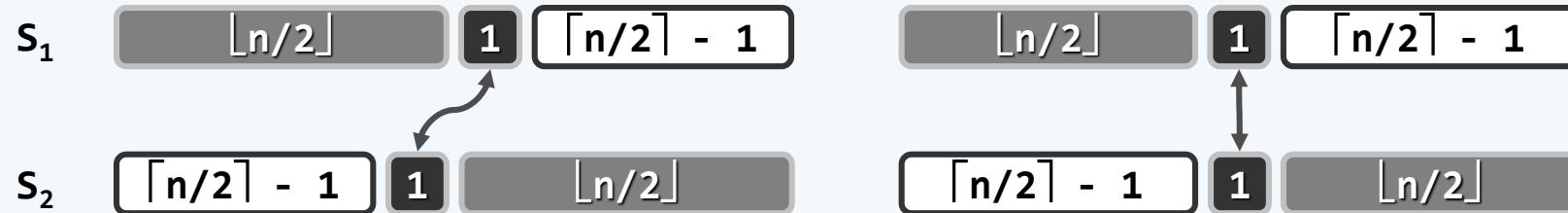
❖ 以下, 先介绍 $|S_1| = |S_2| = n$ 情况下的算法

然后, 再将该算法推广至一般情况

❖ 新的算法, 依然采用减而治之策略...

等长子向量：构思

❖ 考查： $m_1 = S_1[\lfloor n/2 \rfloor]$, $m_2 = S_2[\lceil n/2 \rceil - 1] = S_2[\lfloor (n-1)/2 \rfloor]$



❖ 若 $m_1 = m_2$, 则它们同时是 S_1 、 S_2 和 S 的中位数

❖ 若 $m_1 < m_2$, 则无论 n 为偶为奇, 灰色区间 // $m_1 > m_2$ 同理

或者 不是 S 的中位数; 或者 与 m_1 或 m_2 同为 S 的中位数

这意味着, 剪除 这些区间之后, S 中位数的数值 保持不变

❖ 总之, 每经一次比较, 原问题的规模即 大致减半 —— 整体不过 $\mathcal{O}(\log n)$

等长子向量：实现

❖ template <typename T> //尾递归，可改写为迭代形式

```
T median( Vector<T> & S1, int lo1, Vector<T> & S2, int lo2, int n ) {  
    if ( n < 3 ) return trivialMedian( S1, lo1, n, S2, lo2, n ); //递归基  
    int mi1 = lo1 + n/2, mi2 = lo2 + (n - 1)/2; //长度减半  
    if ( S1[ mi1 ] < S2[ mi2 ] ) //取S1右半、S2左半  
        return median( S1, mi1, S2, lo2, n + lo1 - mi1 );  
    else if ( S1[ mi1 ] > S2[ mi2 ] ) //取S1左半、S2右半  
        return median( S1, lo1, S2, mi2, n + lo2 - mi2 );  
    else  
        return S1[ mi1 ];  
}
```

任意子向量：实现

```
template <typename T>

T median ( Vector<T> & S1, int lo1, int n1, Vector<T> & S2, int lo2, int n2 ) {

    if ( n1 > n2 )

        return median( S2, lo2, n2, S1, lo1, n1 ); //确保n1 <= n2

    if ( n2 < 6 )

        return trivialMedian( S1, lo1, n1, S2, lo2, n2 ); //递归基

    if ( 2 * n1 < n2 )

        return median( S1, lo1, n1, S2, lo2 + (n2-n1-1)/2, n1+2-(n2-n1)%2 );
```

任意子向量：实现

```
int mi1 = lo1 + n1/2, mi2a = lo2 + (n1 - 1)/2, mi2b = lo2 + n2 - 1 - n1/2;

if ( S1[ mi1 ] > S2[ mi2b ] ) //取S1左半、S2右半

    return median( S1, lo1, n1 / 2 + 1, S2, mi2a, n2 - (n1 - 1) / 2 );

else if ( S1[ mi1 ] < S2[ mi2a ] ) //取S1右半、S2左半

    return median( S1, mi1, (n1 + 1) / 2, S2, lo2, n2 - n1 / 2 );

else //S1保留，S2左右同时缩短

    return median( S1, lo1, n1, S2, mi2a, n2 - (n1 - 1) / 2 * 2 );

} // $\mathcal{O}(\log(\min(n1, n2)))$ ——可见，实际上等长版本才是难度最大的
```

12. 排序

选取：通用算法

世兄的才名，弟所素知的。在世兄是数万人里头
选出来最清最雅的，至于弟乃庸庸碌碌一等愚人，
忝附同名，殊觉玷辱了这两个字。

邓俊辉

deng@tsinghua.edu.cn

尝试：蛮力

❖ 对A排序 // $\theta(n \log n)$

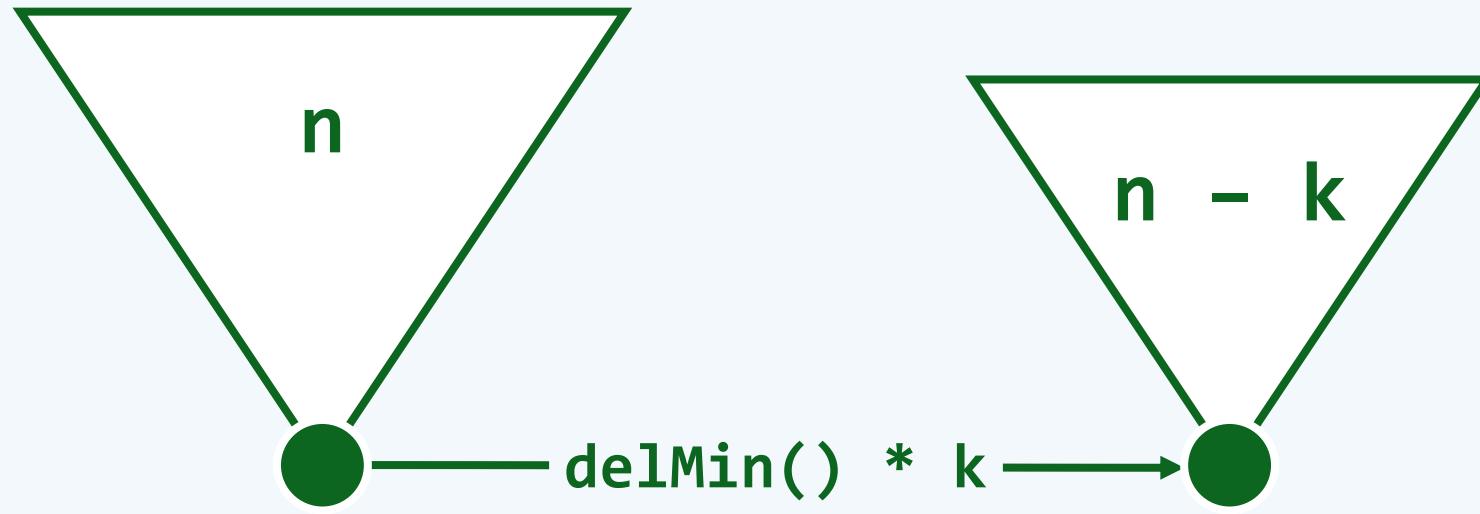
从前向后行进k步 // $\theta(k) = \theta(n)$



尝试：堆 (A)

❖ 将所有元素组织为小顶堆 $// O(n)$

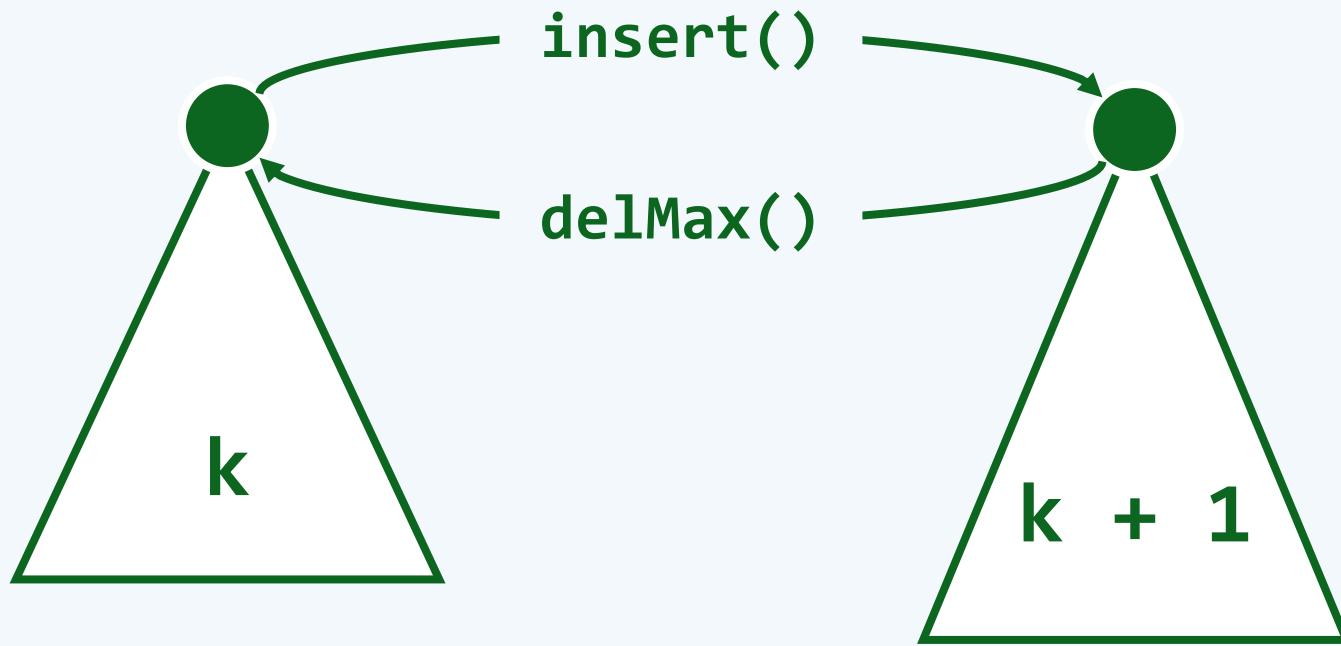
连续调用 k 次 `delMin()` $// O(k \log n)$



尝试：堆 (B)

❖ 任选（比如前） k 个元素，组织为大顶堆 // $\Theta(k)$

对于剩余的 $n - k$ 个元素，各调用一次`insert()`和`delMax()` // $\Theta(2*(n - k)*\log k)$



尝试：堆 (c)

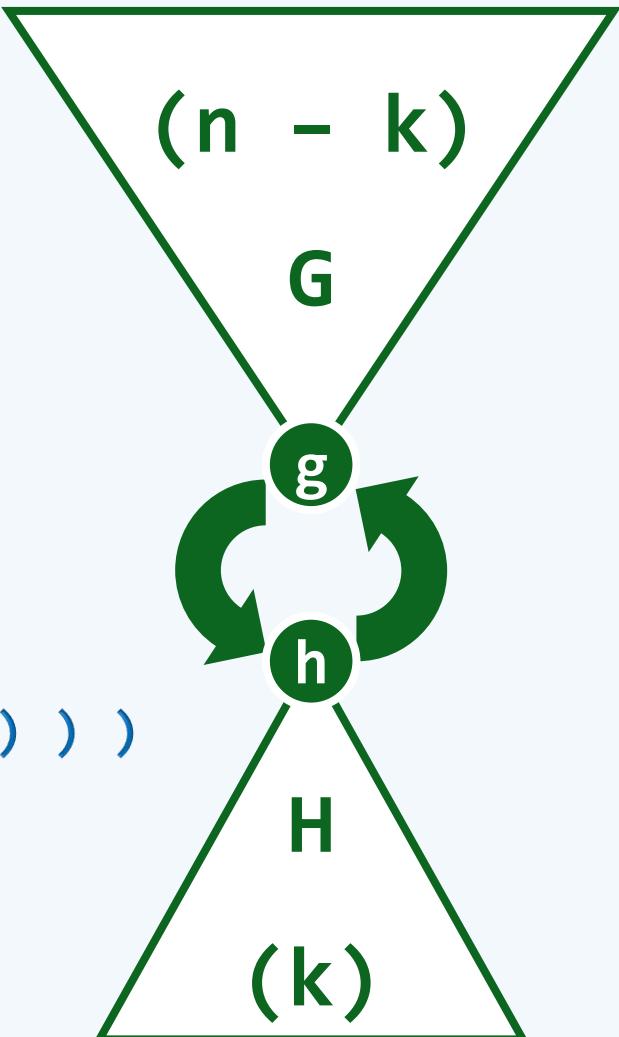
❖ **H**: 任取 k 个元素，组织为 **大顶堆** $\text{// } \mathcal{O}(k)$

G: 其余 $n - k$ 个元素，组织为 **小顶堆** $\text{// } \mathcal{O}(n - k)$

❖ 反复地： 比较 **h** 和 **g** $\text{// } \mathcal{O}(1)$

如有必要，**交换之** $\text{// } \mathcal{O}(2 \times (\log k + \log(n - k)))$

直到： **$h \leq g$** $\text{// } \mathcal{O}(\min(k, n - k))$



尝试：计数排序



- ❖ 是否存在更快的算法?
- ❖ $\Omega(n)!$
- ❖ 所谓第k大，是相对于序列整体而言
在访问每个元素至少一次之前，绝无可能确定
- ❖ 反过来，是否存在 $O(n)$ 的算法?

BFPRT 算法

JOURNAL OF COMPUTER AND SYSTEM SCIENCES 7, 448–461 (1973)

Time Bounds for Selection*

MANUEL BLUM, ROBERT W. FLOYD, VAUGHAN PRATT,
RONALD L. RIVEST, AND ROBERT E. TARJAN

Department of Computer Science, Stanford University, Stanford, California 94305

Received November 14, 1972

作者里面有几位图灵奖得主？

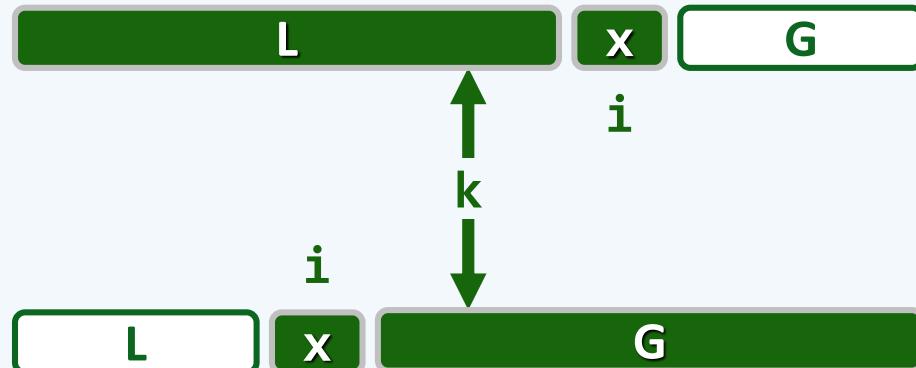
PICK: (Selects $i \in S$, where $|S| = n$ and $1 \leq i \leq n$)

1. (Select an element $m \in S$):
 - (a) Arrange S into n/c columns of length c , and sort each column.
 - (b) Select $m = b \in T$, where $T = \{x \mid x \text{ is the } d\text{-th smallest element from each column}\}$. Use PICK recursively if $n/c > 1$.
2. (Compute $m \rho S$): Compare m to every other element x in S for which it is not yet known whether $m < x$ or $m > x$.
3. (Discard or halt): If $m \rho S = i$, halt (since $m = i \in S$), otherwise if $m \rho S > i$, discard $D = \{x \mid x \geq m\}$ and set $n \leftarrow n - |D|$, otherwise discard $D = \{x \mid x \leq m\}$ and set $n \leftarrow n - |D|$, $i \leftarrow i - |D|$.

Return to step 1.

quickSelect()

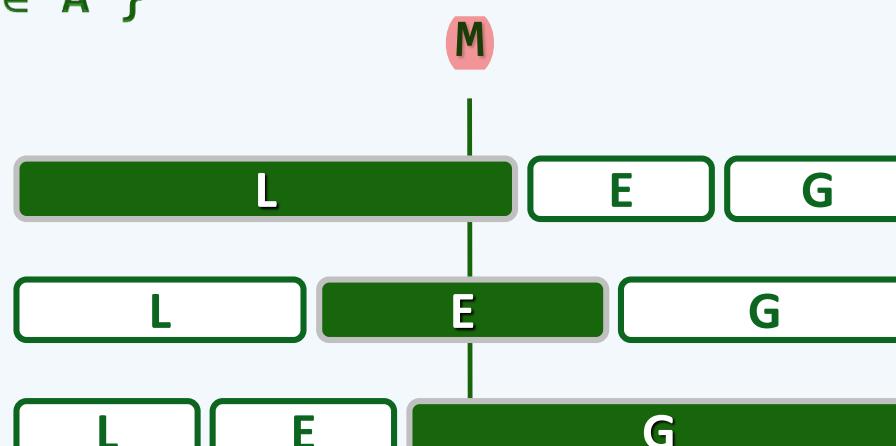
```
template <typename T> void quickSelect( Vector<T> & A, Rank k ) {  
    for ( Rank lo = 0, hi = [A.size() - 1]; lo < hi; ) {  
        Rank i = lo, j = hi; T pivot = A[lo];  
        while ( [i < j] ) { //O(hi - lo + 1) = O(n)  
            while ( [i < j] && [pivot <= A[j]] ) j--; A[i] = A[j];  
            while ( [i < j] && [A[i] <= pivot] ) i++; A[j] = A[i];  
        } //assert: i == j  
        A[i] = pivot;  
        if ( [k <= i] ) hi = [i - 1];  
        if ( [i <= k] ) lo = [i + 1];  
    } //A[k] is now a pivot  
}
```



linearSelect()

Let Q be a small constant

0. if ($n = |A| < Q$) return trivialSelect(A, k)
1. else divide A evenly into n/Q subsesquences (each of size Q)
2. Sort each subsequence and determine n/Q medians //e.g. by insertionsort
3. Call linearSelect to find M , median of the medians //by recursion
4. Let $L / E / G = \{ x \mid x < M \text{ or } x = M \text{ or } x > M \mid x \in A \}$
5. if ($k \leq |L|$) return linearSelect(L, k)
if ($k \leq |L| + |E|$) return M
return linearSelect($G, k - |L| - |E|$)



复杂度

- ❖ 将`linearSelect()`算法的运行时间记作 $T(n)$
- ❖ 第0步: $\Theta(1) = \Theta(Q \log Q)$ //递归基: 序列长度 $|A| \leq Q$
- ❖ 第1步: $\Theta(n)$ //子序列划分
- ❖ 第2步: $\Theta(n) = \Theta(1) \times n/Q$ //子序列各自排序, 并找到中位数
- ❖ 第3步: $T(\lfloor n/Q \rfloor)$ //从 $\lfloor n/Q \rfloor$ 个中位数中, 递归地找到全局中位数
- ❖ 第4步: $\Theta(n)$ //划分子集 $L/E/G$, 并分别计数 —— 一趟扫描足矣
- ❖ 第5步: $T(\lfloor 3n/4 \rfloor)$ //为什么...

复杂度

❖ 在某种意义上，如上所确定的 M 必然 不偏不倚

至少各有 $n/4$ 个元素， 不小于 / 不大于 M

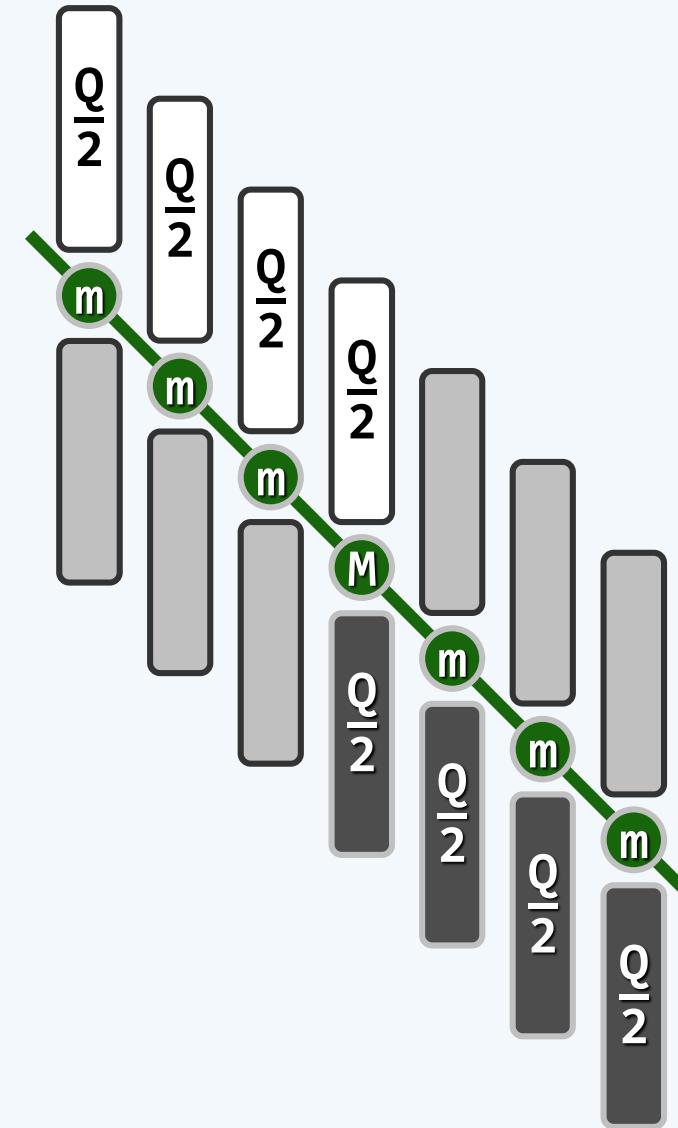
❖ n/Q 个中位数中， 至少半数 不小于 M

而它们又各自不大于至少 $Q/2$ 个元素

$$\frac{n}{Q} / 2 * \frac{Q/2}{2} = \frac{n}{4}$$

❖ $\min(|L|, |G|) + |E| \geq n/4$

$\max(|L|, |G|) \leq 3n/4$



复杂度

❖ $T(n) = \Theta(n) + T(n/Q) + T(3n/4)$

❖ 为使之解作线性函数，只需保证

$$\frac{n}{Q} + \frac{3n}{4} < n$$

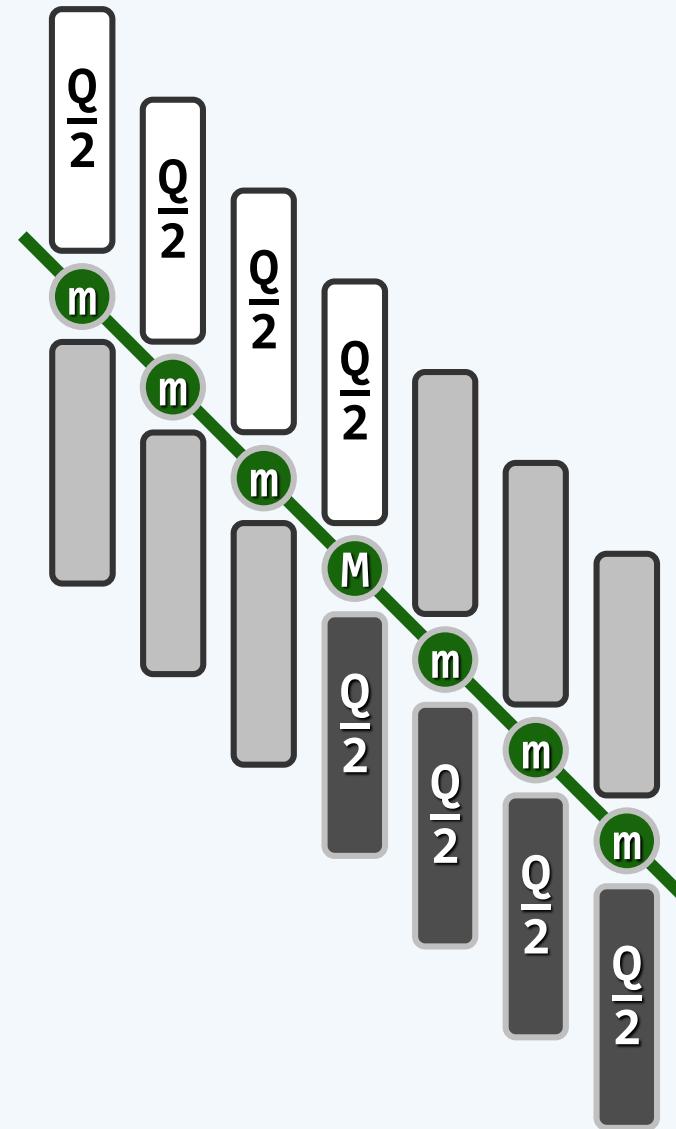
或等价地

$$\frac{1}{Q} + \frac{3}{4} < 1$$

❖ 比如，若取 $Q = 5$ ，则存在常数 c ，使得

$$T(n) = cn + T(n/5) + T(3n/4)$$

$$T(n) = \Theta(20cn) = \Theta(n)$$



Next

- Review