14-A1

排序

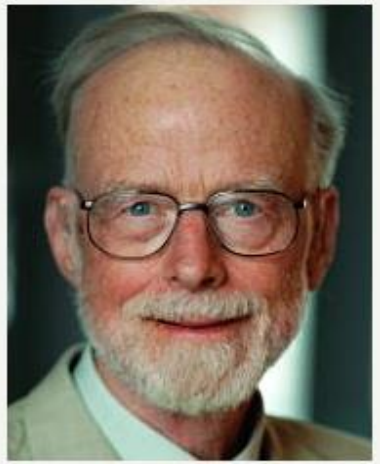
快速排序：轴点

邓俊辉

deng@tsinghua.edu.cn

左朱雀之芨芨兮，右苍龙之躍躍

分而治之



C. A. R. Hoare
(1934 ~)
Turing Award, 1980

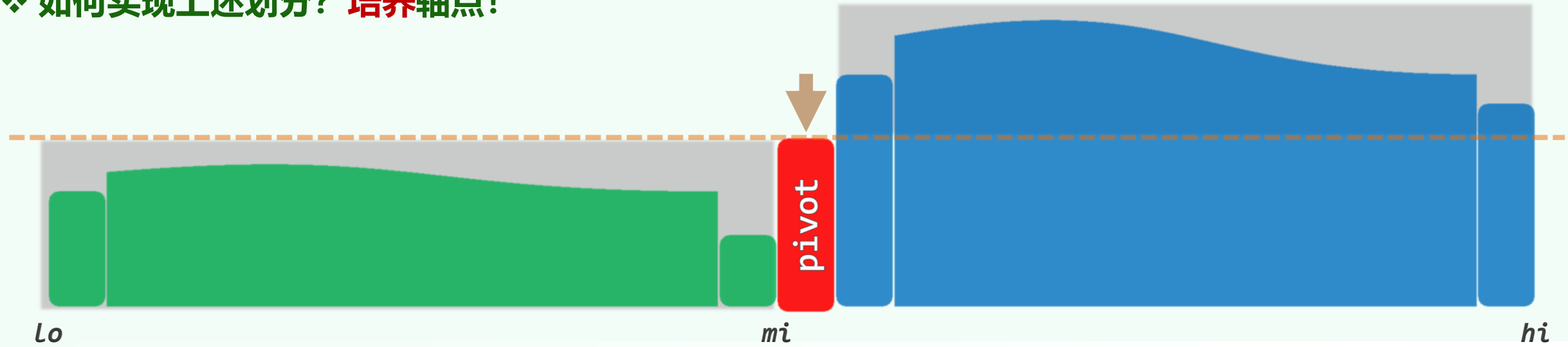
❖ pivot: $\max[lo, mi) \leq [mi] \leq \min(mi, hi)$

❖ 前缀、后缀各自（递归）排序之后，原序列便自然有序

$$sorted(S) = sorted(S_L) + pivot + sorted(S_R)$$

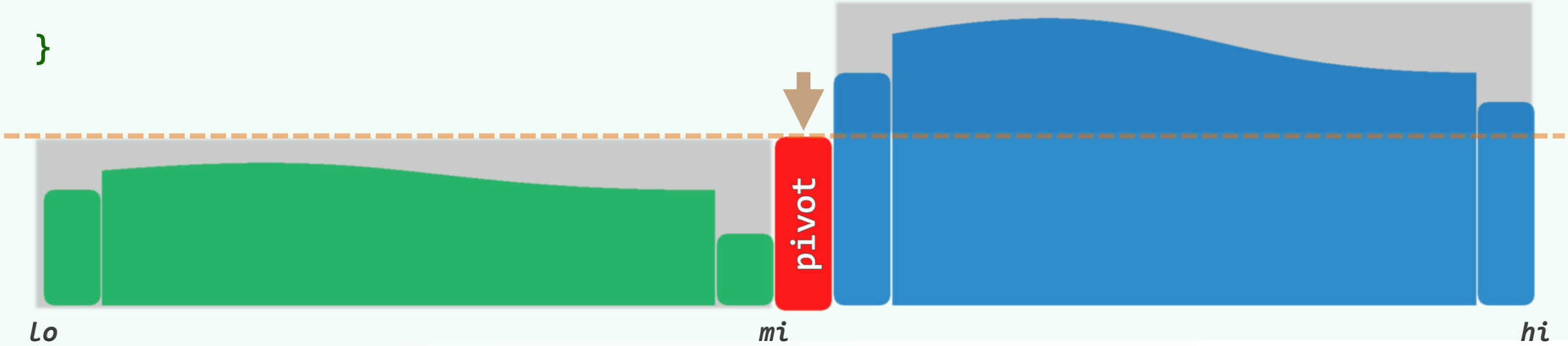
❖ mergesort难点在于合，而quicksort在于分

❖ 如何实现上述划分？培养轴点！



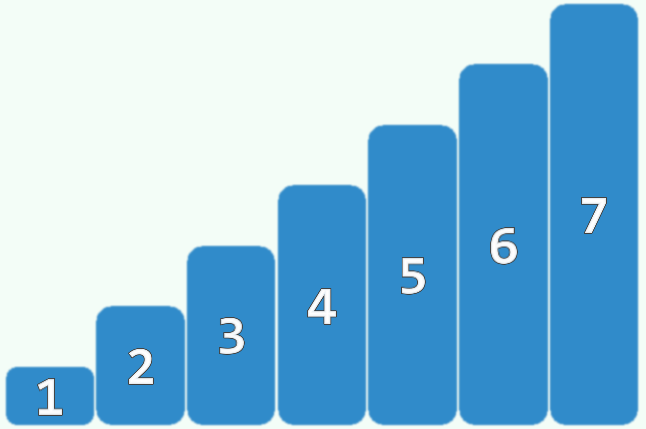
快速排序

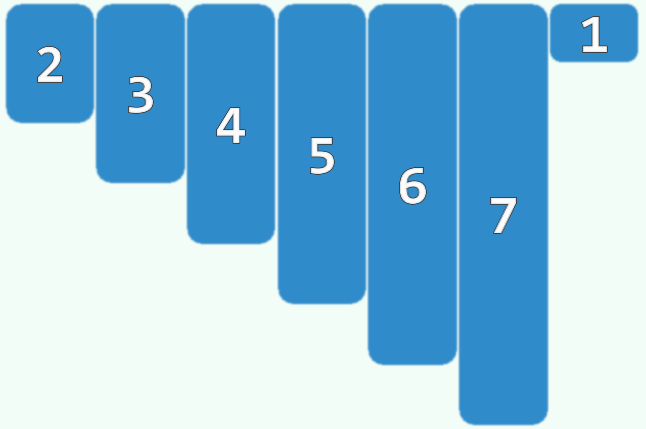
```
template <typename T> void Vector<T>::quickSort( Rank lo, Rank hi ) {
   if ( hi - lo < 2 ) return;
   Rank mi = partition( lo, hi ); //能否足够高效?
   quickSort( lo, mi );
   quickSort( mi + 1, hi );
}
```



轴点

- 必要条件：轴点必定已然就位 //尽管反之不然
- 进一步地：一个序列有序，当且仅当所有元素皆为轴点
- 快速排序：就是将所有元素逐个转换为轴点的过程
- 坏消息：一个序列中未必总有轴点...也就是所谓的 derangement：任何元素都不在原位
  比如，顺序序列循环移位
- 好消息：不需很多交换，即可使任一元素转为轴点
- 问题：如何交换？成本多高？





14-A2

排序

快速排序：快速划分：LUG版

邓俊辉
deng@tsinghua.edu.cn

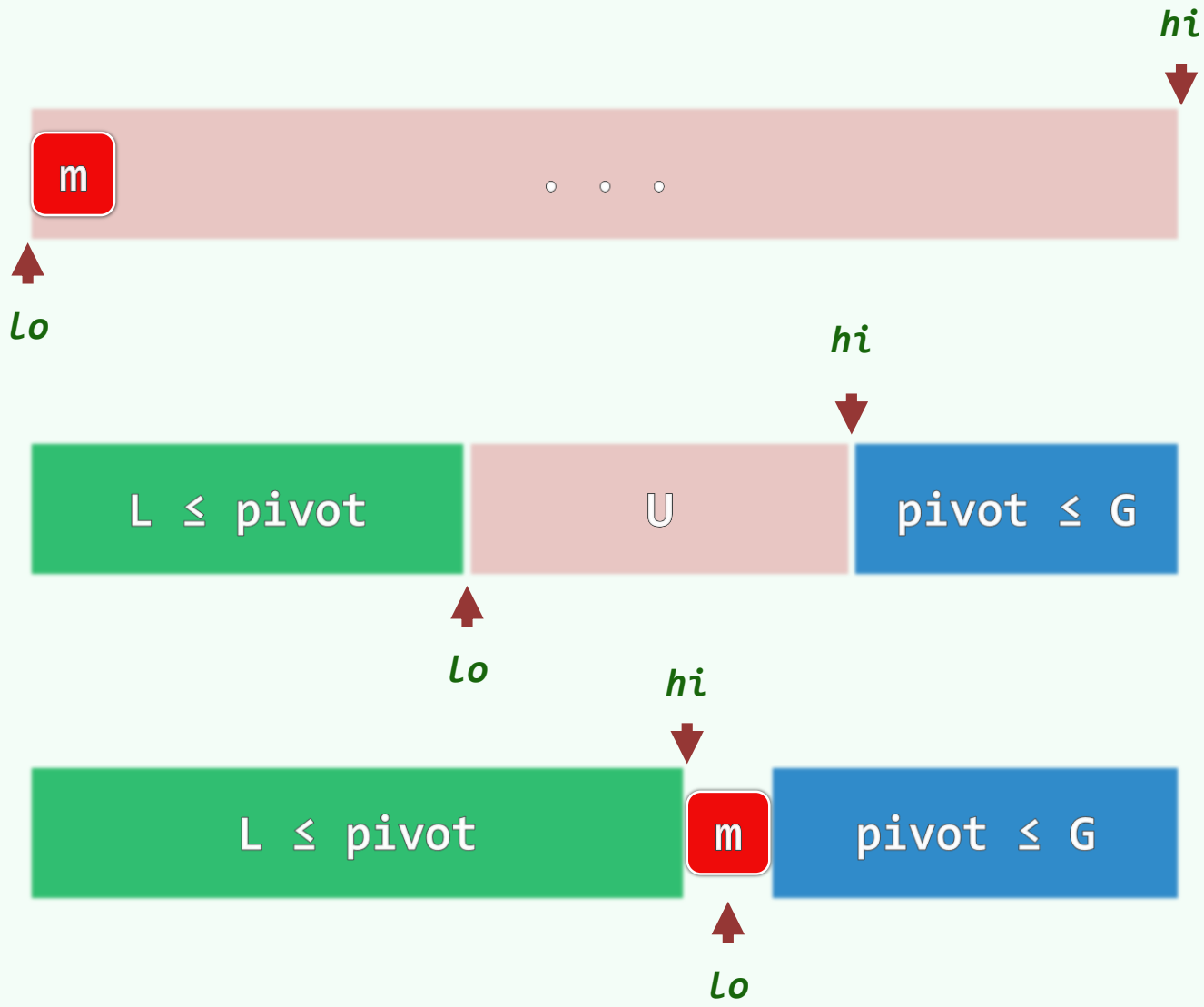
减而治之，相向而行

- 任取一个候选者（如[0]）
- L + U + G
- 交替地向内移动lo和hi
- 逐个检查当前元素：

  若更小/大，则转移归入L/G
- 当lo = hi时，只需

  将候选者嵌入于L、G之间，即成轴点！
- 各元素最多移动一次（候选者两次）

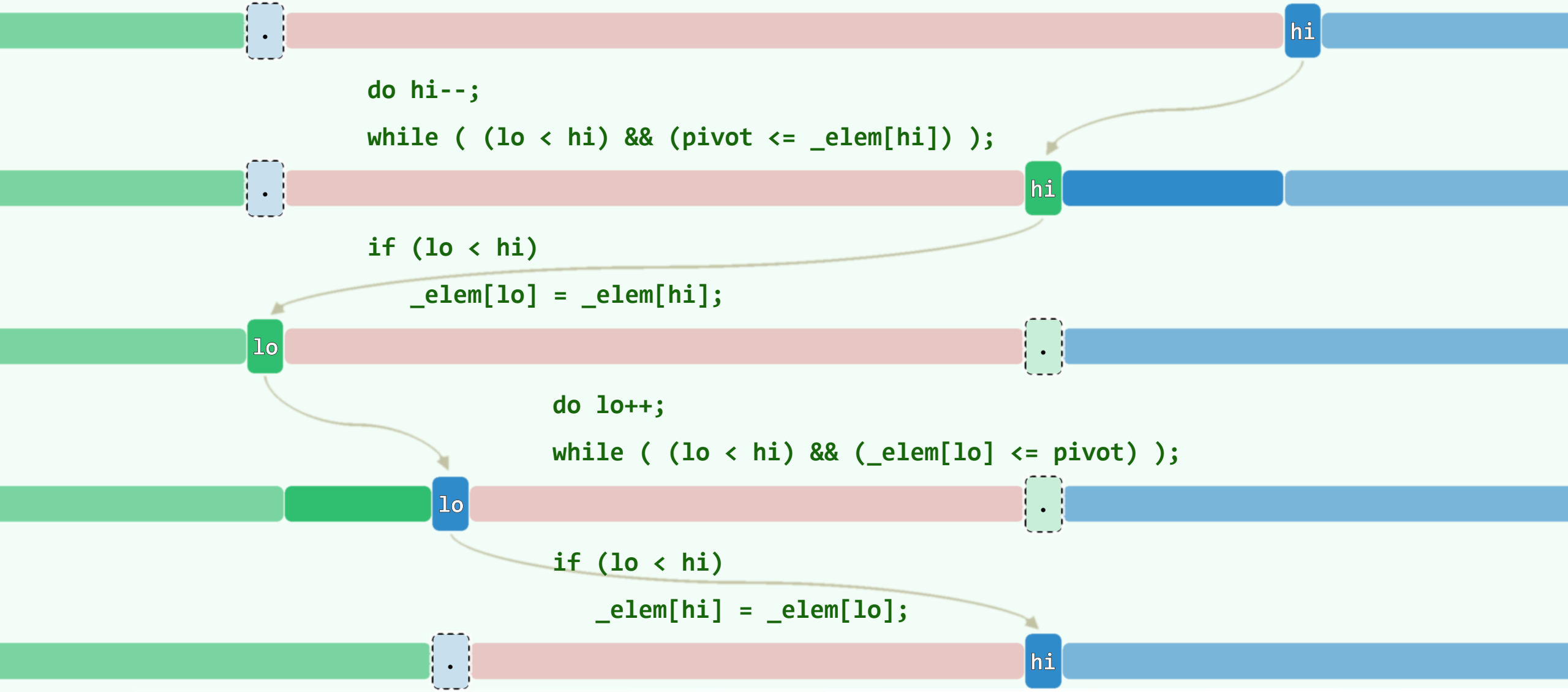
  ——累计O(n)时间、O(1)辅助空间



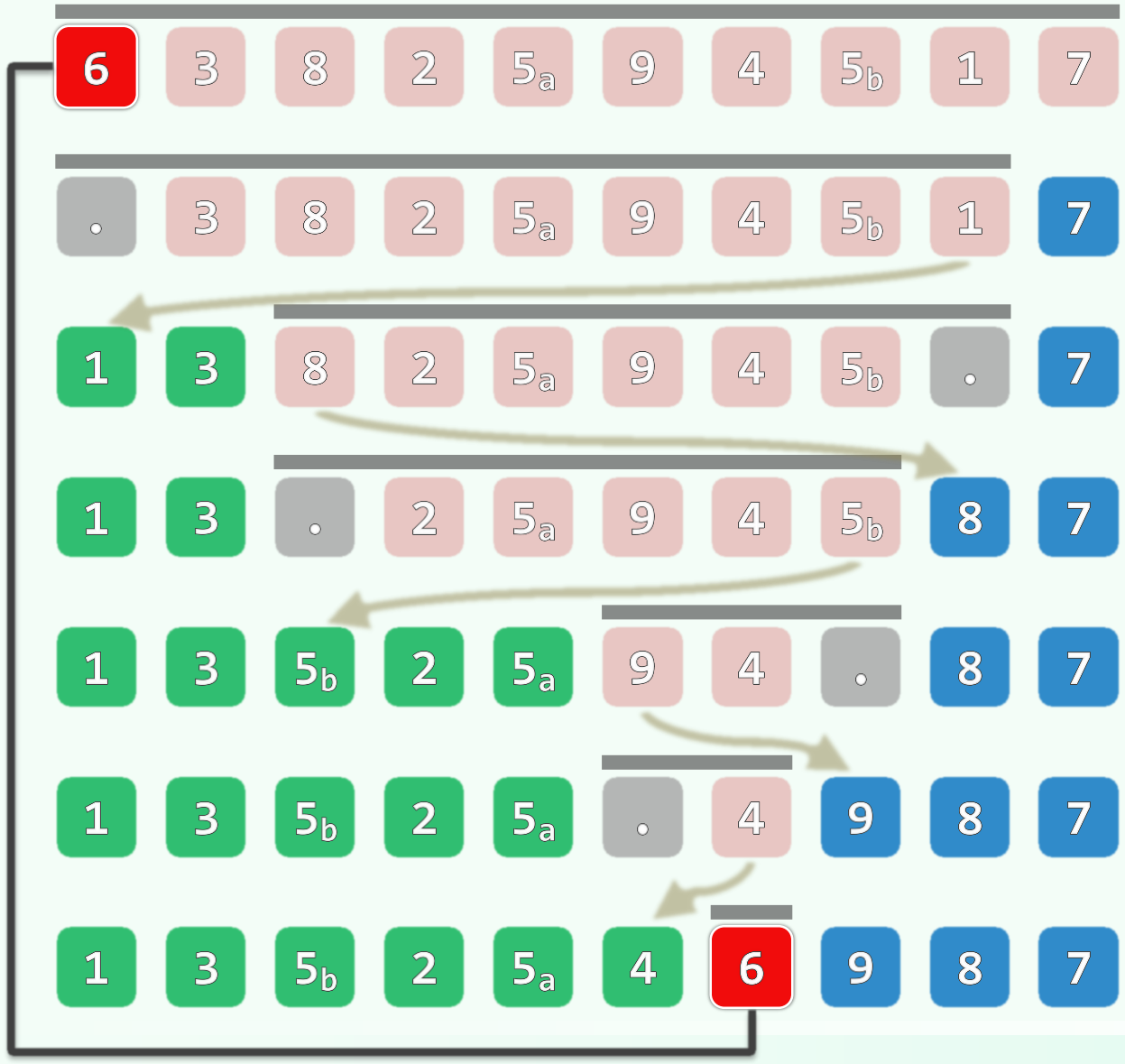
快速划分：LUG版

```
template <typename T> Rank Vector<T>::partition( Rank lo, Rank hi ) { //[lo, hi)
   swap( _elem[lo], _elem[lo + rand()%(hi-lo)] ); T pivot = _elem[lo]; //随机轴点

   while ( lo < hi ) { //从两端交替地向中间扫描，彼此靠拢
      do hi--; while ( (lo < hi) && (pivot <= _elem[hi]) ); //向左拓展G
      if (lo < hi) _elem[lo] = _elem[hi]; //凡 小于 轴点者，皆归入L
      do lo++; while ( (lo < hi) && (_elem[lo] <= pivot) ); //向右拓展L
      if (lo < hi) _elem[hi] = _elem[lo]; //凡 大于 轴点者，皆归入G
   } //assert: lo == hi or hi+1

   _elem[hi] = pivot; return hi; //候选轴点归位；返回其秩
}
```

不变性: L = [0,lo); U = (lo,hi); G = [hi,n); [lo] == [hi]



实例



- ❖ 线性时间
  - 尽管lo、hi交替移动
  - 累计移动距离不过$O(n)$
- ❖ 就地/in-place
  - 只需$O(1)$附加空间
- ❖ unstable
  - lo/hi的移动方向相反
  - 相等的元素，可能前/后颠倒

14-A3

排序

快速排序：迭代、贪心与随机

察一叶而知天下秋

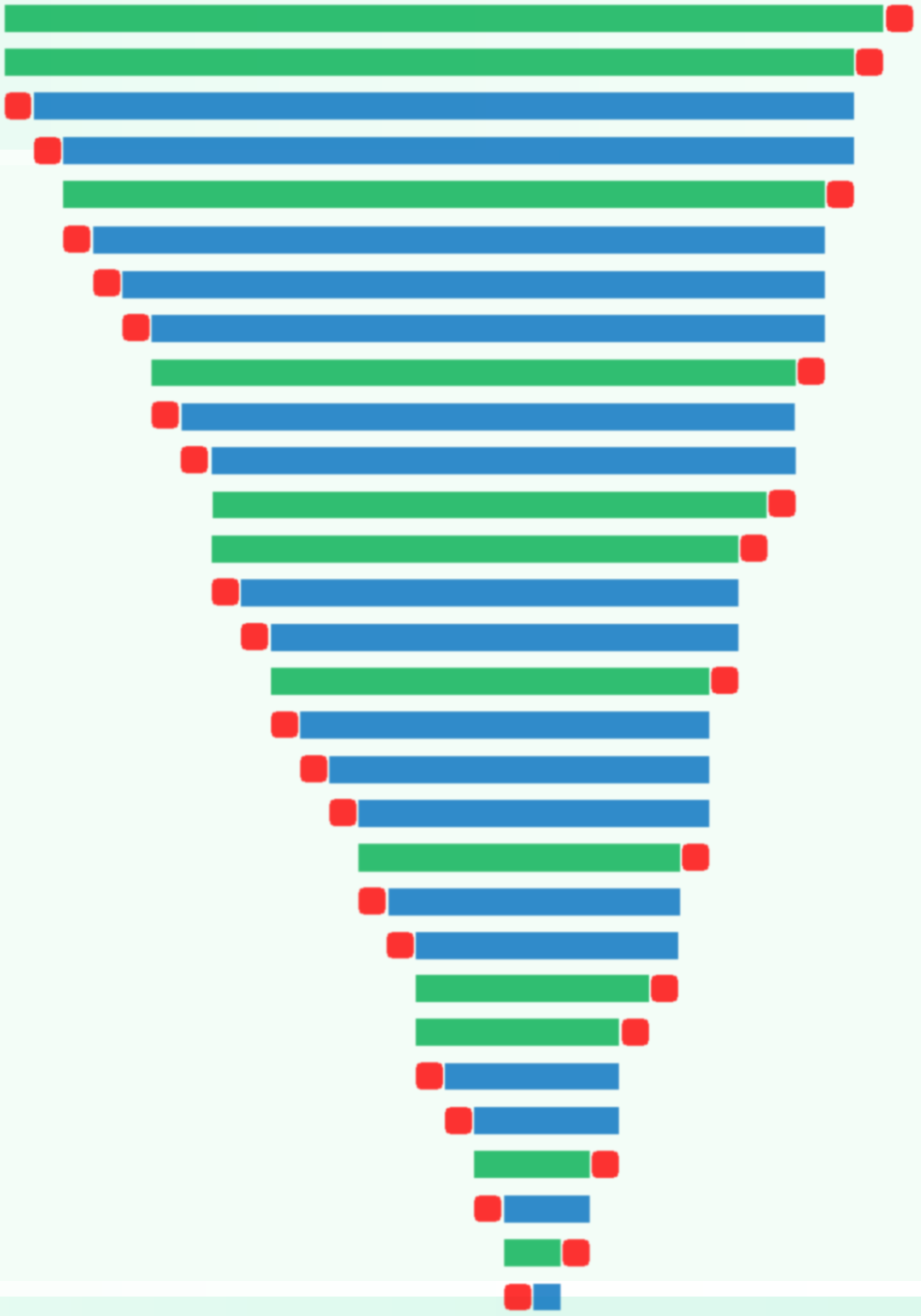
搬不搬瑜，瑜不搬搬，忠也

邓俊辉

deng@tsinghua.edu.cn

空间复杂度 ~ 递归深度

- 最好：划分总均衡 $\mathcal{O}(\log n)$
  最差：划分皆偏侧 $\mathcal{O}(n)$
  平均：均衡不致太少 $\mathcal{O}(\log n)$
- 可否避免最坏情况？如何避免？

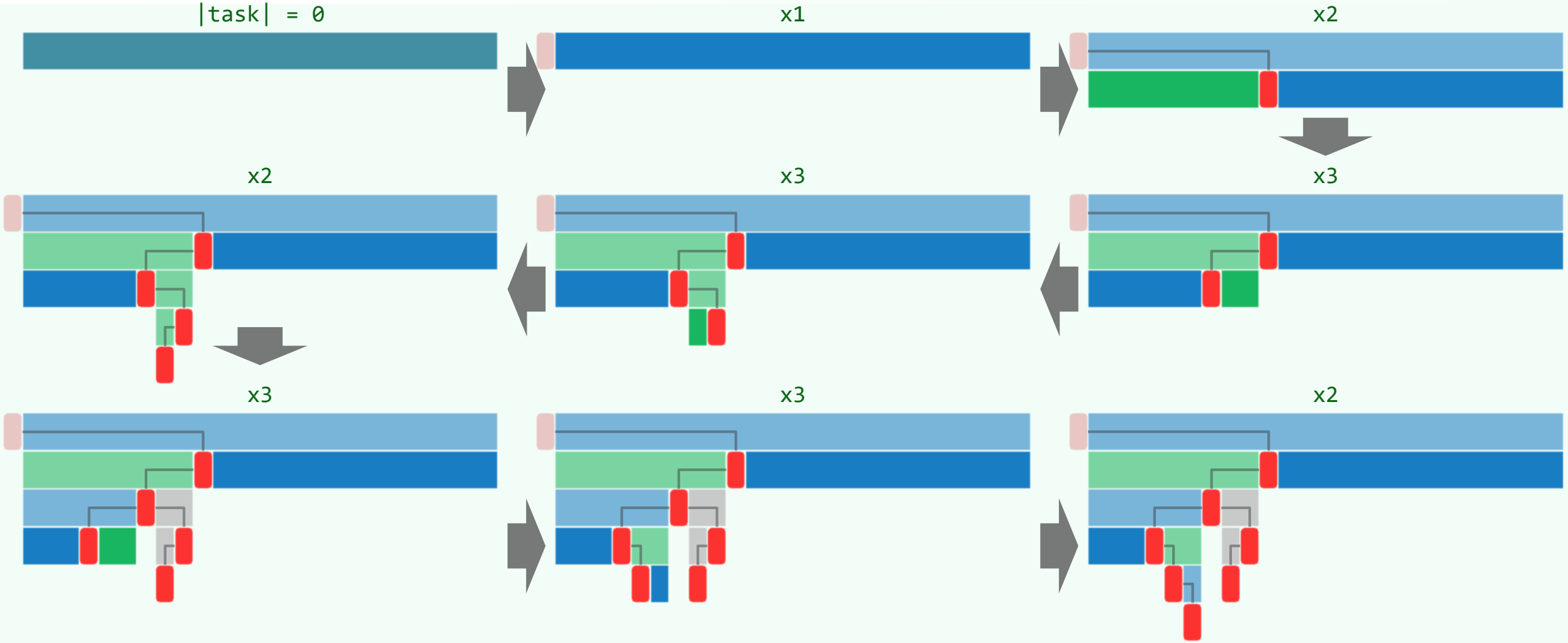


迭代化 + 贪心

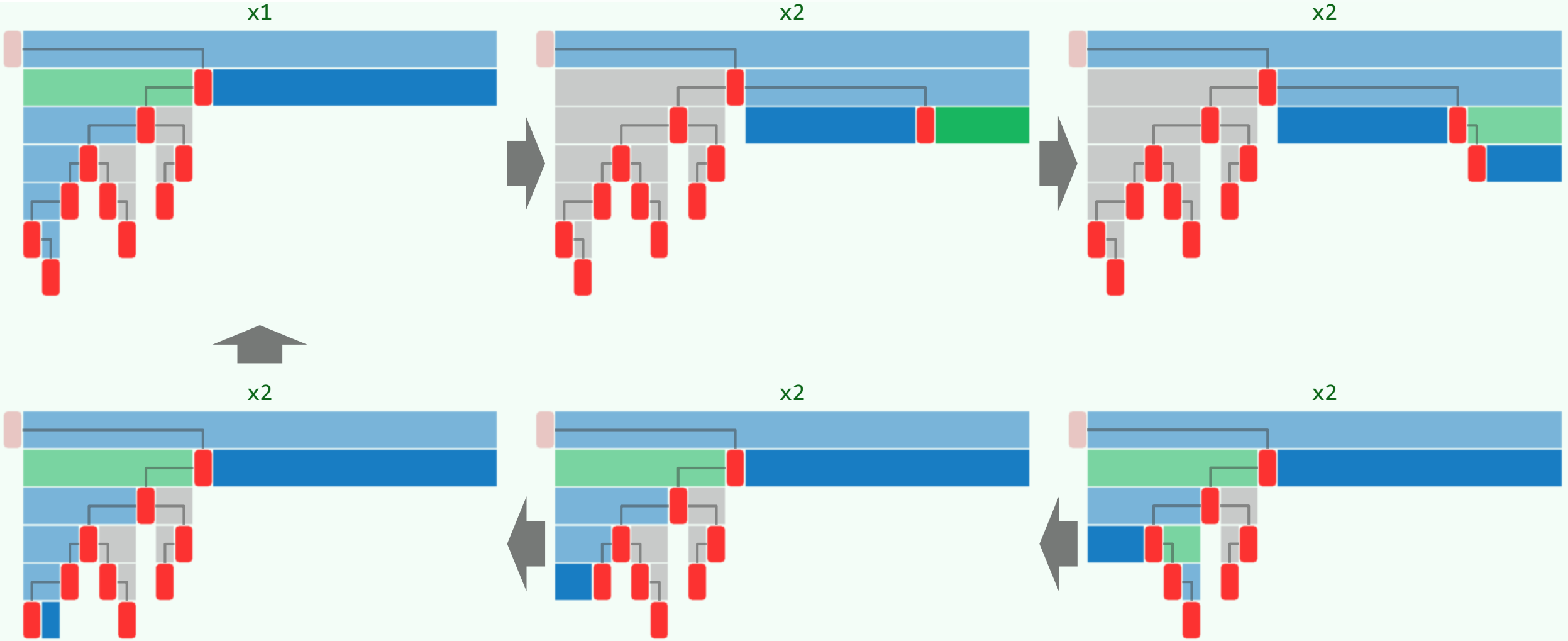
```
#define Put( K, s, t ) { if ( 1 < (t) - (s) ) { K.push(s); K.push(t); } }
#define Get( K, s, t ) { t = K.pop(); s = K.pop(); }
```

```
template <typename T> void Vector<T>::quickSort( Rank lo, Rank hi ) {
   Stack<Rank> Task; Put( Task, lo, hi ); //类似于对递归树的先序遍历
   while ( !Task.empty() ) {
      Get( Task, lo, hi );  Rank mi = partition( lo, hi );
      if ( mi-lo < hi-mi ) { Put( Task, mi+1, hi ); Put( Task, lo, mi ); }
      else                 { Put( Task, lo, mi ); Put( Task, mi+1, hi ); }
   }  //大|小任务优先入|出栈，可保证（辅助栈）空间不过O(logn)
}
```

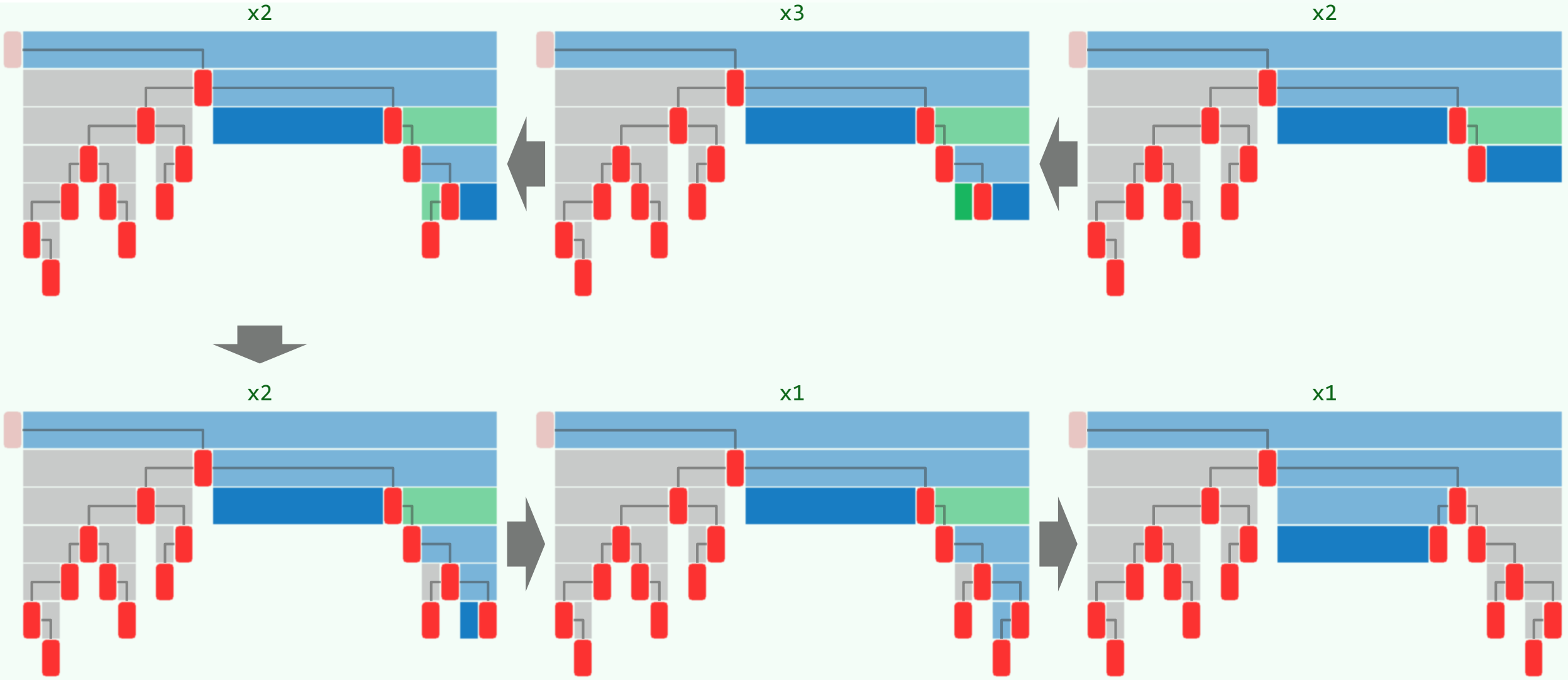
实例：0 ~ 7



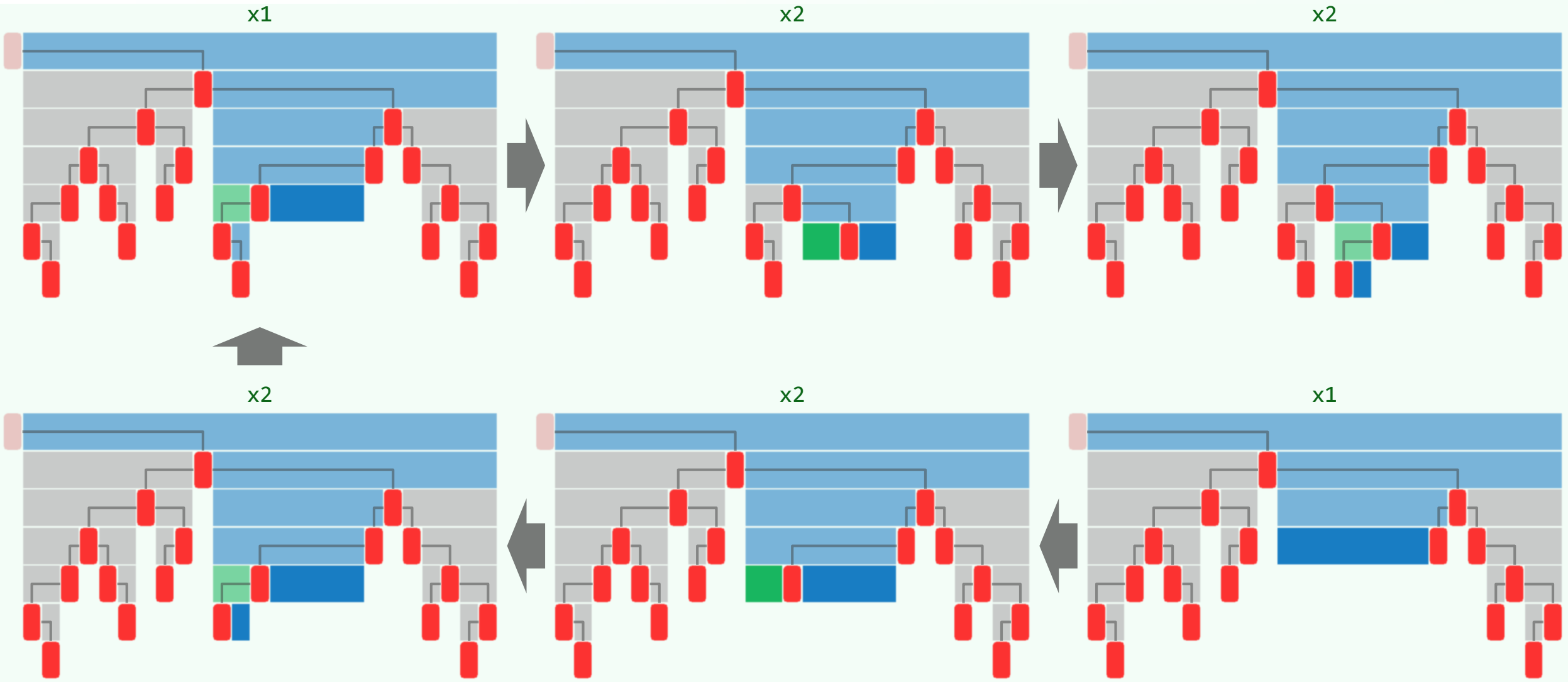
实例：7 ~ 12



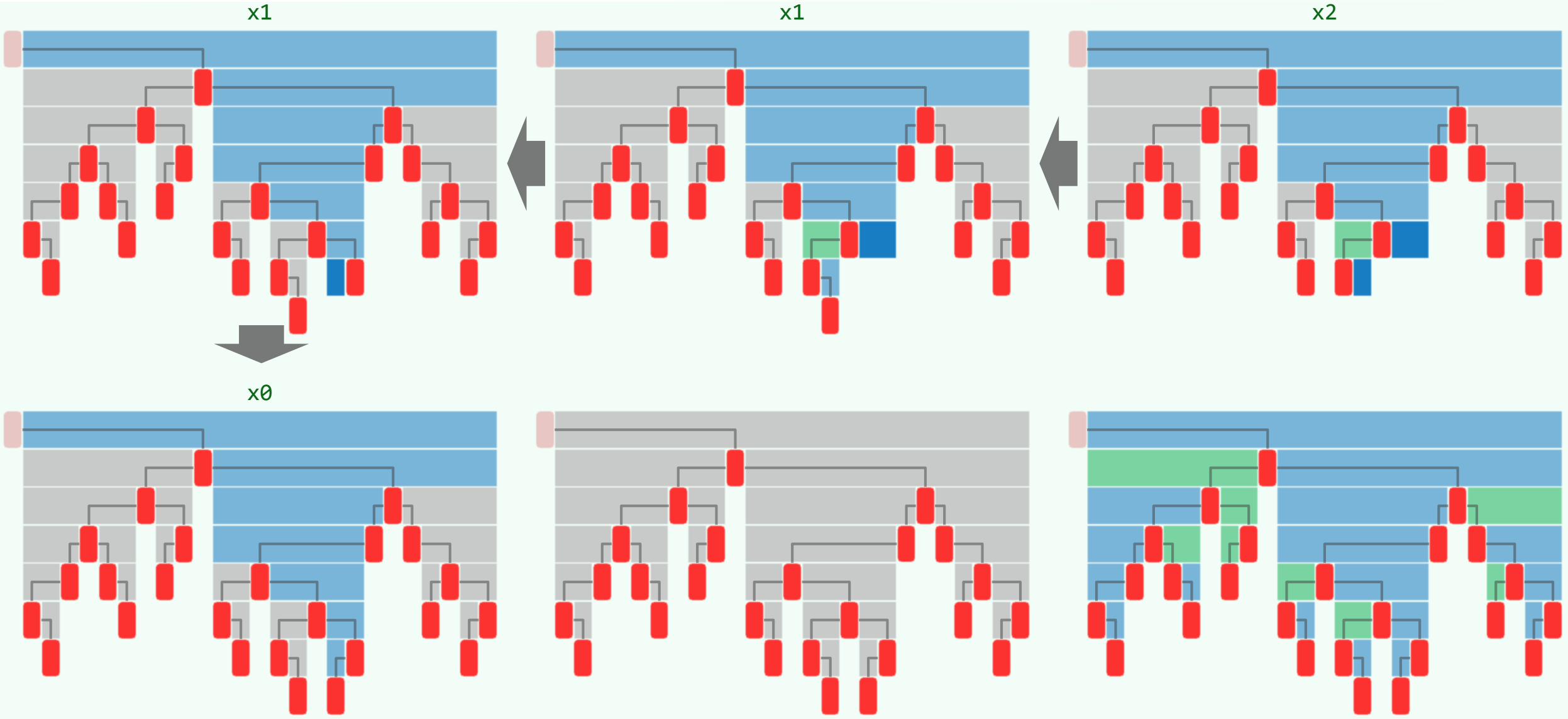
实例：12 ~ 17



实例：17 ~ 22



实例：22 ~ 25



空间复杂度 $\mathcal{O}(\log n)$

❖ 归纳假设：对长度 $m < n$ 的序列，该算法所需空间不超过 $\log m$

❖ 考查长度为 n 的序列，算法执行过程可分为三个阶段：

X：经过第一次迭代（划分）之后，|Task| = 2

- 栈顶子任务 V 必是轻的：$|V| = v \leq \lfloor n/2 \rfloor$
- 栈底子任务 U 必有削减：$|U| = u \leq n - 1 < n$

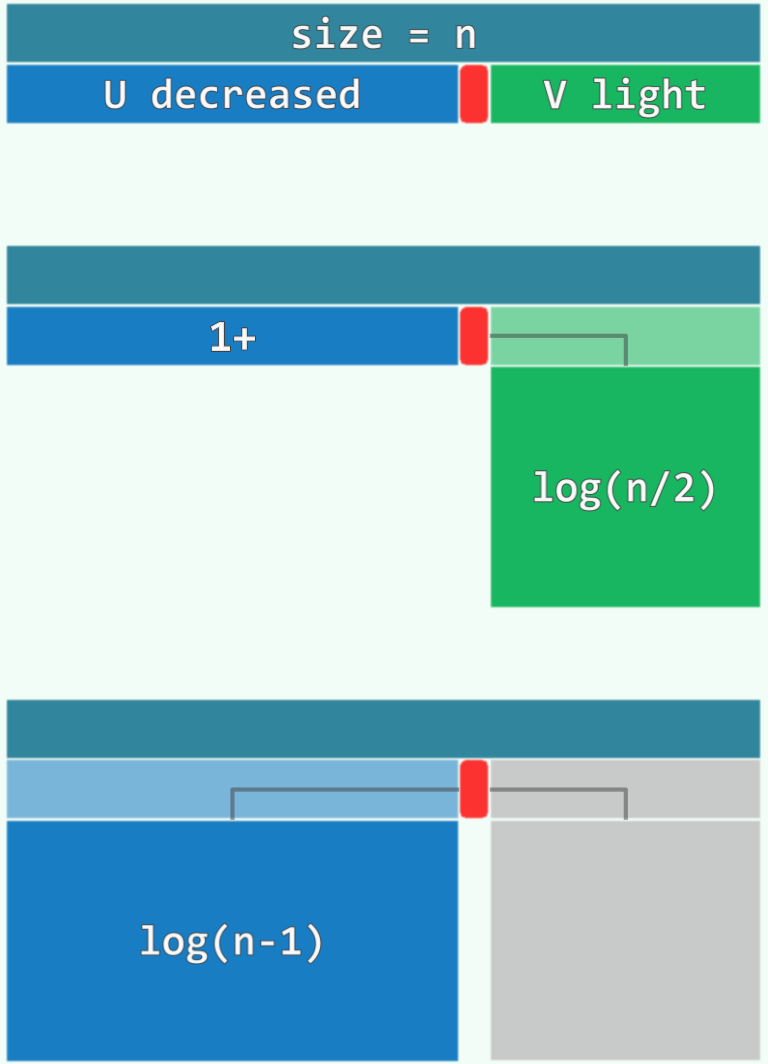
V：接下来，在对 V 的排序（共 v 次划分）过程中

根据归纳假设，算法需要的空间量不超过

$$1 + \log v \leq 1 + \log(n/2) = \log n$$

U：再接下来，在对 U 的排序（共 u 次划分）过程中

同样根据归纳假设，所需的空间量不超过 $\log u < \log n$



时间性能 + 随机

❖ 最好情况：每次划分都（接近）平均，轴点总是（接近）中央

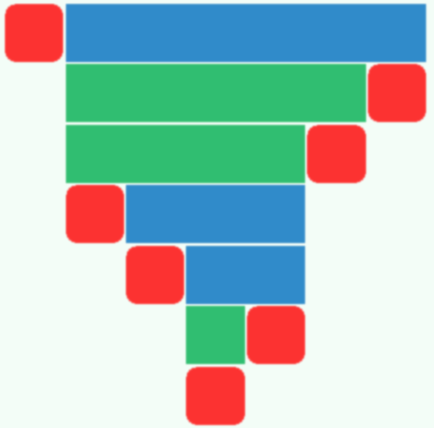
$$T(n) = 2 \cdot T((n-1)/2) + \mathcal{O}(n) = \mathcal{O}(n \cdot \log n)$$

—— 到达下界！

❖ 最坏情况：每次划分都极不均衡（比如，轴点总是最小/大元素）

$$T(n) = T(n-1) + T(0) + \mathcal{O}(n) = \mathcal{O}(n^2)$$

—— 与起泡排序为伍！



❖ 采用随机选取（Randomization）、三者取中（Sampling）之类的策略

只能降低最坏情况的概率，而无法杜绝 —— 既如此，为何还称作快速排序？

14-A4

排序

快速排序：递归深度

楊子拔一毛不爲，墨子又摩頂放踵爲之，此皆是不得中

今夫盲者行于道，人谓之左则左，谓之右则右。遇君子则得其平易，遇小人则蹈于沟壑

邓俊辉
deng@tsinghua.edu.cn

居中 + 偏侧：三者取中（1/2）

❖ **好**轴点：落在宽度为 $\lambda \cdot n$ 的居中区间（概率为 λ ）

坏轴点：落在两侧（概率为 $1-\lambda$ ）

(1-λ)/2	width = λ = Pr.	(1-λ)/2

❖ 若采用**三者取中**策略

以 $\lambda = 0.5$ 为例，**好**轴点的概率为

$$
\begin{aligned}
& 1 \times 0.50 \cdot 0.50 \cdot 0.50 \\
+ \; & 3 \times 0.50 \cdot 0.50 \cdot 0.25 \\
+ \; & 3 \times 0.25 \cdot 0.50 \cdot 0.50 \\
+ \; & 6 \times 0.25 \cdot 0.50 \cdot 0.25 \\
= \; & 68.75\%
\end{aligned}
$$

Pr.	0.25	0.50	0.25
C(3,3) * 0.25*0.25*0.25 = 0.015625	xOx		
C(3,3) * 0.25*0.25*0.25 = 0.015625			xOx
C(3,2)*C(1,1) * 0.25*0.25*0.50 = 0.093750	xO	x	
C(3,1)*C(2,2) * 0.50*0.25*0.25 = 0.093750		x	Ox
C(3,2)*C(1,1) * 0.25*0.25*0.25 = 0.046875	xO		x
C(3,1)*C(2,2) * 0.25*0.25*0.25 = 0.046875	x		Ox
C(3,3) * 0.50*0.50*0.50 = 0.125000		xOx	
C(3,2)*C(1,1) * 0.50*0.50*0.25 = 0.187500		xO	x
C(3,1)*C(2,2) * 0.25*0.50*0.50 = 0.187500	x	Ox	
C(3,1)*C(2,1)*C(1,1) * 0.25*0.50*0.25 = 0.187500	x	O	x

居中 + 偏侧：三者取中（2/2）

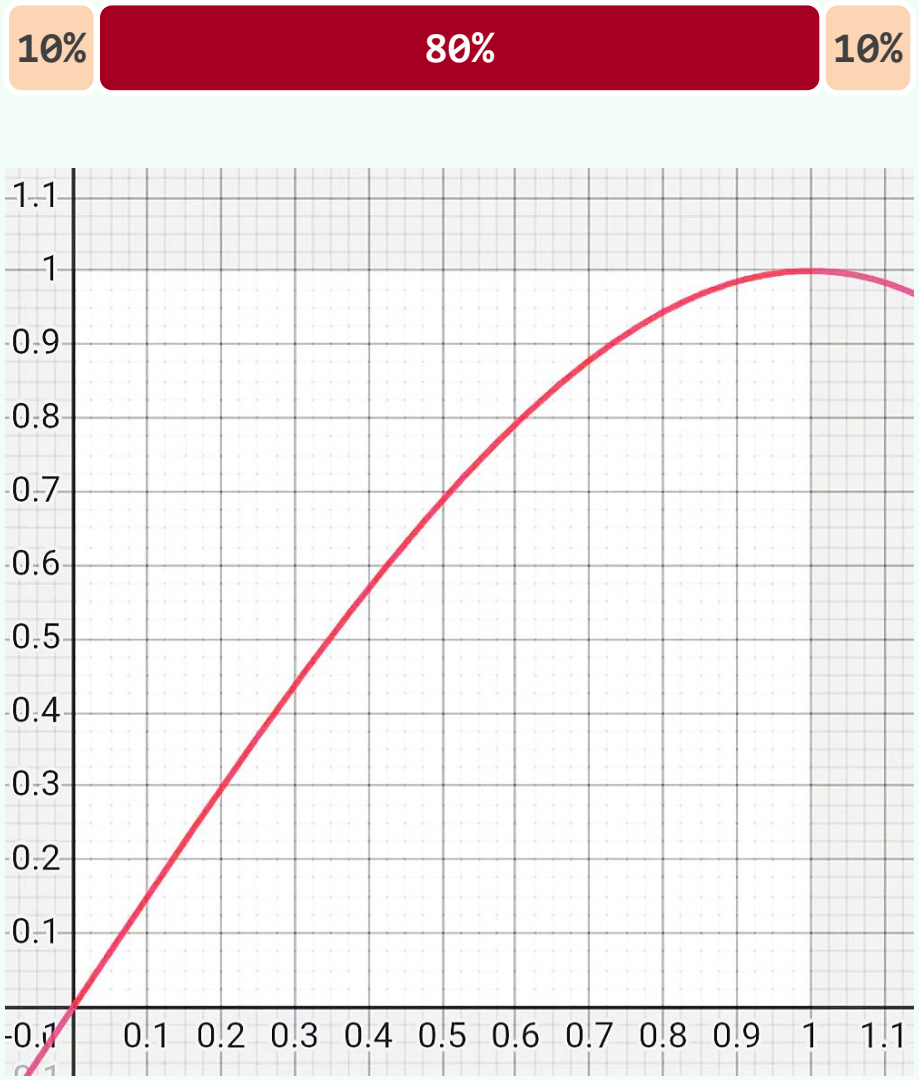
❖ **好**轴点：落在宽度为 $\lambda \cdot n$ 的居中区间（概率为 λ）

坏轴点：落在两侧（概率为 $1-\lambda$）

❖ 若采用**三者取中**策略

以 $\lambda = 0.8$ 为例，**好**轴点的概率为

$$
\begin{aligned}
& 1 \times 0.80 \cdot 0.80 \cdot 0.80 \\
+ \; & 3 \times 0.80 \cdot 0.80 \cdot 0.10 \\
+ \; & 3 \times 0.10 \cdot 0.80 \cdot 0.80 \\
+ \; & 6 \times 0.10 \cdot 0.80 \cdot 0.10 \\
= \; & 94.40\%
\end{aligned}
$$



递归深度：常规随机

❖ 最坏情况递归 $\Omega(n)$ 层，概率极低

平均情况递归 $\mathcal{O}(\log n)$ 层，概率极高

❖ 实际上：除非过于侧偏的pivot，

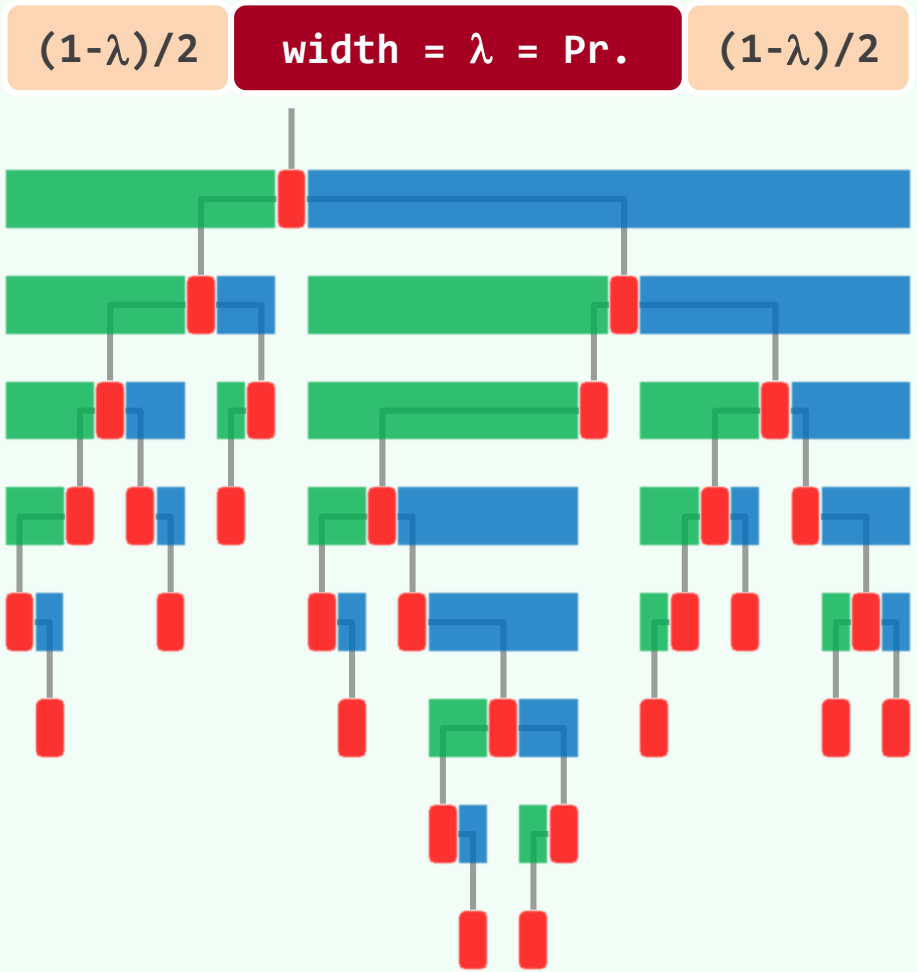
都会有效地缩短递归深度

❖ 断言：在任何一条递归路径上，好轴点绝不会多于

$$d(n,\lambda) = \log_{2/(1+\lambda)} n$$

❖ 断言：抵达 $1/\lambda \cdot d(n,\lambda)$ 层时，即可

期望地出现 $d(n,\lambda)$ 个好轴点 —— 从而在此之前终止递归



递归深度：以 λ=0.5 估计

❖ $d(n,1/2) = \log_{2/(1+1/2)} n = \log_{4/3} n \approx 2.41 \cdot \log n$

❖ 断言：任何一条递归路径的长度，只有极小的概率超过

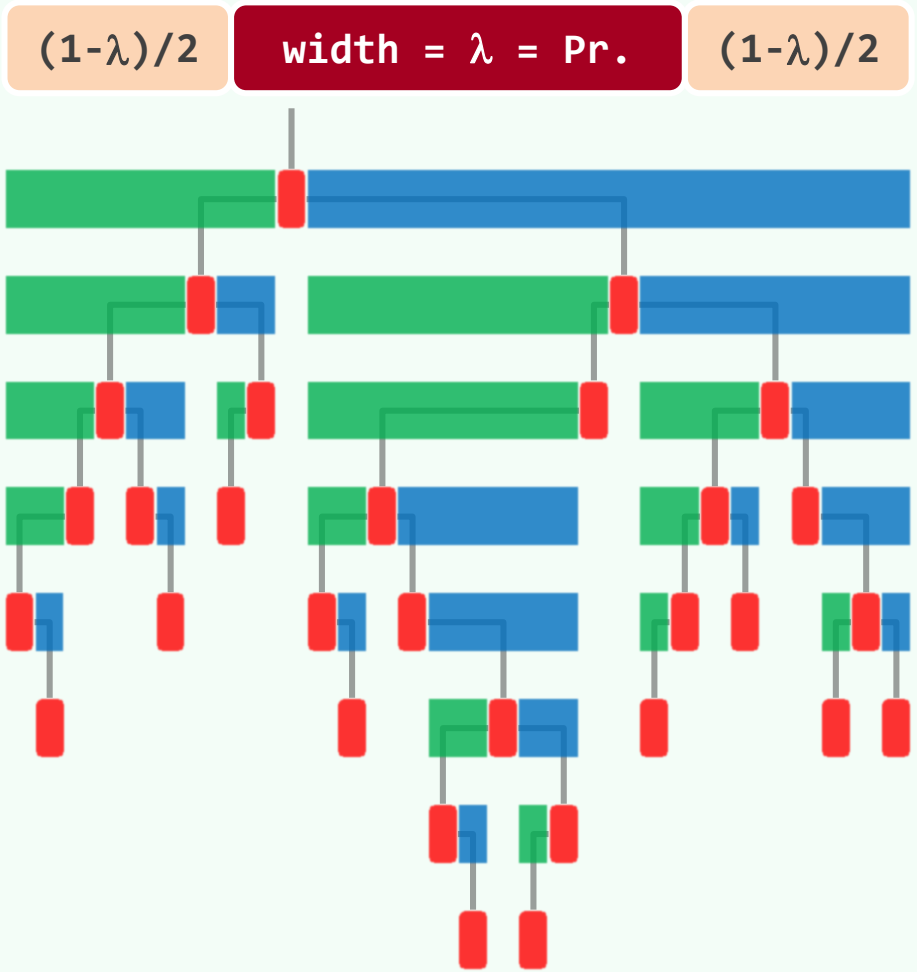
$$D(n,1/2) = 2/\lambda \cdot d(n,1/2) \approx 9.64 \cdot \log_2 n$$

❖ 事实上，此概率

$$\leq \sum_{i=0}^{d} \binom{D}{i} \cdot \lambda^i \cdot (1-\lambda)^{D-i} = 2^{-D} \cdot \sum_{i=0}^{d} \binom{D}{i}$$

$$\leq 2^{-4d} \cdot (eD/d)^d \quad (\because \sum_{i=0}^{k} \binom{N}{i} \leq (eN/k)^k)$$

$$= 16^{-d} \cdot (4e)^d = (e/4)^{\log_{4/3} n} = n^{\log_{4/3} e/4} \approx n^{-1.343}$$



❖ 当 $n = 10^6$ 时，递归深度不超过 D 的概率 $\geq 1 - n^{-0.343} > 99.1223\%$ //occurring w.h.p.

排序

快速排序：比较次数

14-A5

时间究竟是什么
假使人家不问我，我
像很明了
假使要我解释起来，我就茫无头绪

庇拉尔·特尔内拉在这场造梦运动中出力最多，她成功地
将纸牌算命从推演未来应用到追溯过往

邓俊辉
deng@tsinghua.edu.cn

递推分析（1/2）

❖ 记期望的比较次数为 $T(n)$： $T(1)=0,\ T(2)=1,\ \ldots$

❖ 可以证明：$T(n)=\mathcal{O}(n\log n)$ ...

$$T(n) \;=\; (n-1)+\frac{1}{n}\cdot\sum_{k=0}^{n-1}[T(k)+T(n-k-1)] \;=\; (n-1)+\frac{2}{n}\cdot\sum_{k=0}^{n-1}T(k)$$

$$n\cdot T(n) \;=\; n\cdot(n-1) \;+\; 2\times\sum_{k=0}^{n-1}T(k)$$

$$(n-1)\cdot T(n-1) \;=\; (n-1)\cdot(n-2) \;+\; 2\times\sum_{k=0}^{n-2}T(k)$$

递推分析（2/2）

$$n \cdot T(n) - (n-1) \cdot T(n-1) = 2 \cdot (n-1) + 2 \times T(n-1)$$

$$n \cdot T(n) - (n+1) \cdot T(n-1) = 2 \cdot (n-1)$$

$$\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{4}{n+1} - \frac{2}{n}$$

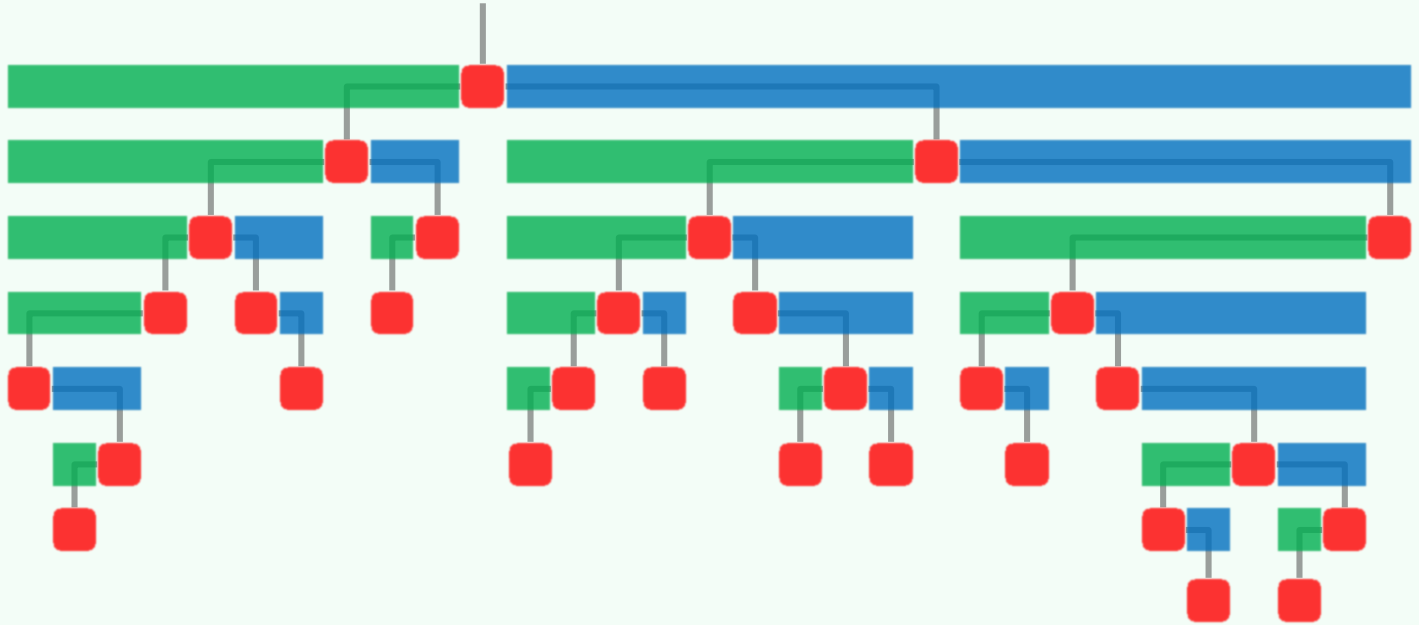
$$\frac{T(n)}{n+1} = \frac{T(n)}{n+1} - \frac{T(1)}{2} = 4 \cdot \sum_{k=2}^{n} \frac{1}{k+1} - 2 \cdot \sum_{k=2}^{n} \frac{1}{k} = 2 \cdot \sum_{k=1}^{n+1} \frac{1}{k} + \frac{2}{n+1} - 4 \approx 2 \cdot \ln n$$

$$T(n) \approx 2 \cdot n \cdot \ln n = (2 \cdot \ln 2) \cdot n \log n \approx 1.386 \cdot n \log n$$

后向分析（1/2）

❖ 设经排序后得到的输出序列为：

$$\{ a_0, a_1, a_2, \cdots\cdots, a_i, \cdots\cdots, a_j, \cdots\cdots, a_{n-1} \}$$



❖ 这一输出与具体使用何种算法无关

故可使用Backward Analysis

❖ 比较操作的期望次数应为

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} Pr(i,j) = \sum_{j=1}^{n-1} \sum_{i=0}^{j-1} Pr(i,j)$$

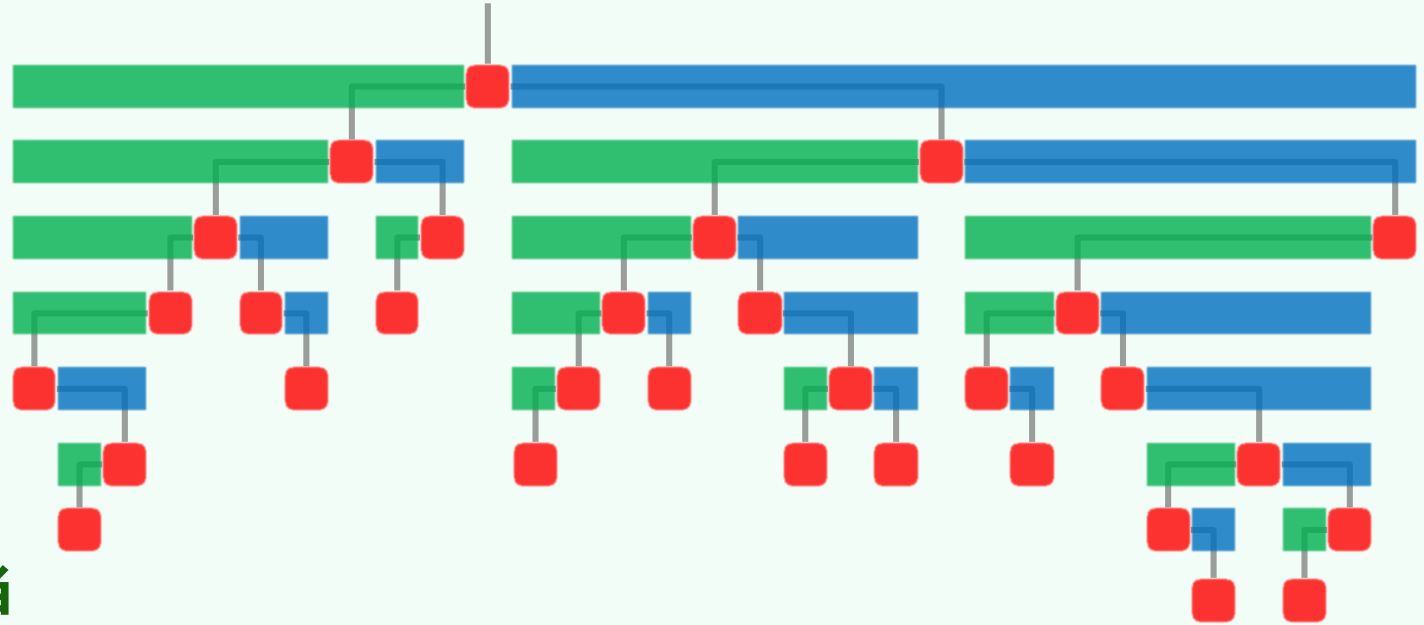
亦即，每一对 $\langle a_i, a_j \rangle$ 在排序过程中会做比较之概率的总和

后向分析（2/2）

❖ quickSort的过程及结果可理解为：

将所有元素逐个地转化为pivot

$$\{\ a_0,\ a_1,\ a_2,\ \ldots\ldots,\ a_i,\ \ldots\ldots,\ a_j,\ \ldots\ldots,\ a_{n-1}\ \}$$



❖ 若 $k \notin [i, j]$，则

a_k 早于或晚于 a_i 和 a_j 被转化，

均与 $Pr(i, j)$ 无关

❖ 进一步地，$\langle a_i, a_j \rangle$ 会做比较，当且仅当

在 $\{\ a_i,\ a_{i+1},\ a_{i+2},\ \ldots,\ a_{j-2},\ a_{j-1},\ a_j\ \}$ 中，a_i 或 a_j 率先被转化

$$T(n) = \sum_{j=1}^{n-1}\sum_{i=0}^{j-1} Pr(i,j) = \sum_{j=1}^{n-1}\sum_{d=1}^{j}\frac{2}{d+1} \approx \sum_{j=1}^{n-1} 2\cdot(\ln j - 1) \leq 2\cdot n\cdot \ln n$$

对比

Vector sorter	#compare	#move	cache-friendly
Insertionsort	$\mathcal{O}(n) \sim \mathcal{O}(n^2)$ expected-$\mathcal{O}(n^2)$	$\mathcal{O}(n) \sim \mathcal{O}(n^2)$ expected-$\mathcal{O}(n^2)$	
Selectionsort	$\Theta(n^2)$	$\mathcal{O}(n)$	
Heapsort	2.0*nlogn	1.0*nlogn	percolateDown(): ✗[i] ✗[j]
Mergesort	1.0*nlogn	1.5*nlogn	merge(): ✓[i] ✓[j] ✓[k]
Quicksort	expected- 1.386*nlogn w.h.p.	expected- (1.386/2 = 0.694)*nlogn	✓✓pivot ✓[lo] ✓[hi]

排序

快速排序：快速划分：DUP版

14-A6

邓俊辉

deng@tsinghua.edu.cn

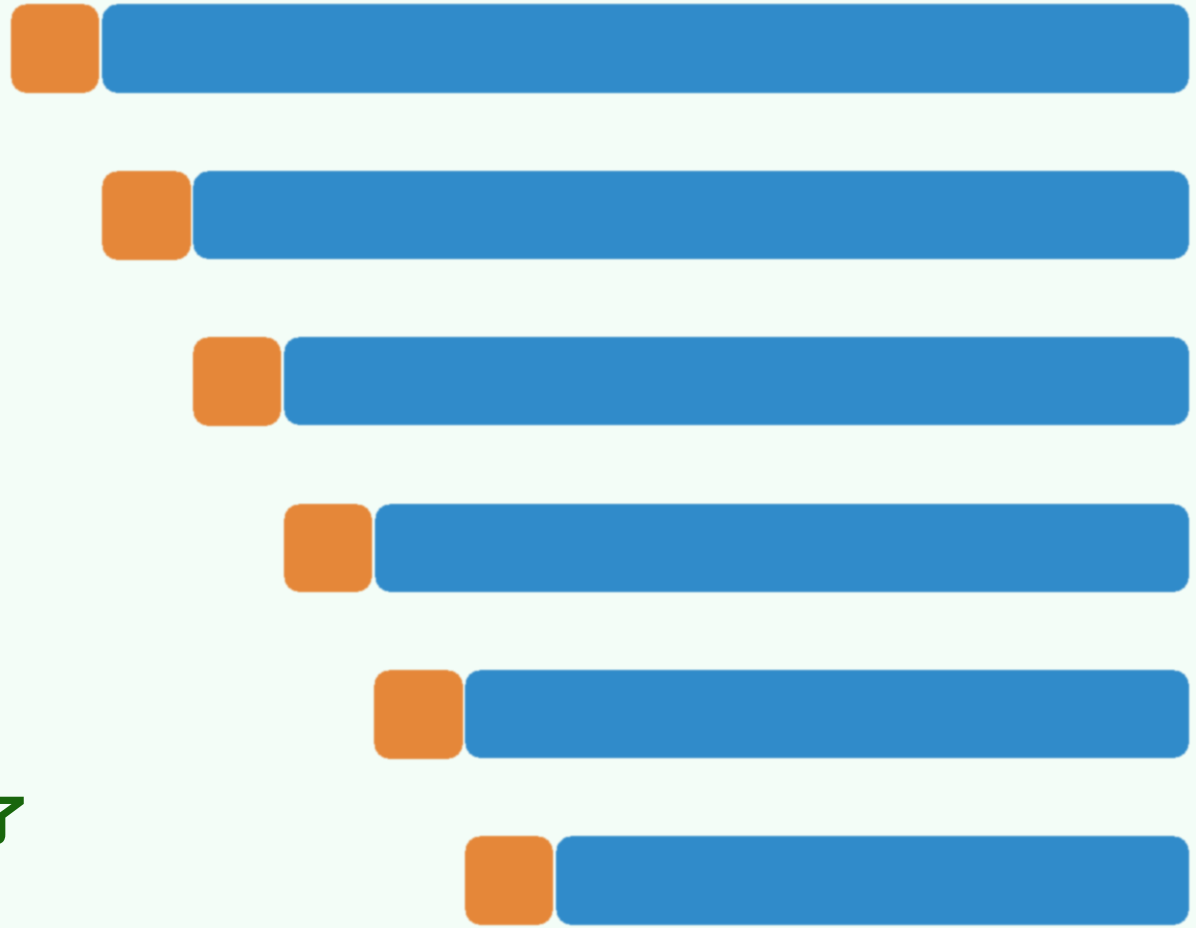
那么，一个生命的出生也就是一个世界的出生了，任何个人，都是独一无二的世界

相等元素

- ❖ 有大量元素与轴点相等时
  - 切分点将接近于lo

    划分极度失衡
  - 递归深度接近于$O(n)$

    运行时间接近于$O(n^2)$
  - 而尴尬的是...
- ❖ 当所有元素都相等时，其实已经无需排序了
- ❖ 实际上，LUG版略做调整，即可解决问题...



快速划分：LUG版

```
template <typename T> Rank Vector<T>::partition( Rank lo, Rank hi ) { //[lo, hi)
   swap( _elem[lo], _elem[lo + rand()%(hi-lo)] ); T pivot = _elem[lo]; //随机轴点

   while ( lo < hi ) { //从两端交替地向中间扫描，彼此靠拢
      do hi--; while ( (lo < hi) && (pivot <= _elem[hi]) ); //向左拓展G
      if (lo < hi) _elem[lo] = _elem[hi]; //凡 小于 轴点者，皆归入L
      do lo++; while ( (lo < hi) && (_elem[lo] <= pivot) ); //向右拓展L
      if (lo < hi) _elem[hi] = _elem[lo]; //凡 大于 轴点者，皆归入G
   } //assert: lo == hi or hi+1

   _elem[hi] = pivot; return hi; //候选轴点归位；返回其秩
}
```

快速划分：DUP版

```
template <typename T> Rank Vector<T>::partition( Rank lo, Rank hi ) { //[lo, hi)
   swap( _elem[lo], _elem[lo + rand()%(hi-lo)] ); T pivot = _elem[lo]; //随机轴点

   while ( lo < hi ) { //从两端交替地向中间扫描，彼此靠拢
      do hi--; while ( (lo < hi) && (pivot <  _elem[hi]) ); //向左拓展G
      if (lo < hi) _elem[lo] = _elem[hi]; //凡不大于轴点者，皆归入L
      do lo++; while ( (lo < hi) && (_elem[lo] <  pivot) ); //向右拓展L
      if (lo < hi) _elem[hi] = _elem[lo]; //凡不小于轴点者，皆归入G
   } //assert: lo == hi or hi+1

   _elem[hi] = pivot; return hi; //候选轴点归位；返回其秩
}
```

性能

- 可以正确地处理一般情况

  同时复杂度并未实质增高

- 遇到连续的相等元素时
  - lo和hi会交替移动
  - 切分点接近于(lo+hi)/2
- 由LUG版的勤于拓展、懒于交换

  转为懒于拓展、勤于交换
- 交换操作有所增加，且更不稳定

14-A7

排序

快速排序：快速划分：LGU版

The Great walks with the Small without fear.

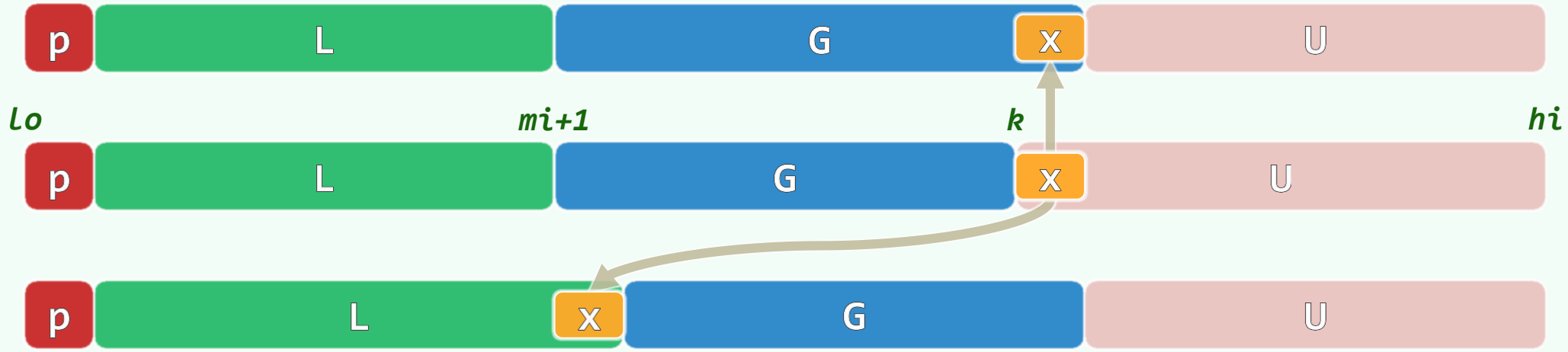
The Middle keeps aloof.

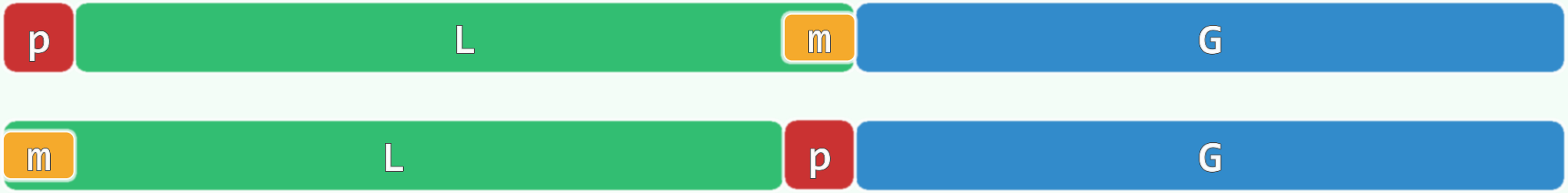
邓俊辉

deng@tsinghua.edu.cn

不变性： $S = [lo, hi) = [lo] + (lo, mi] + (mi, k) + [k, hi) = pivot + L + G + U$

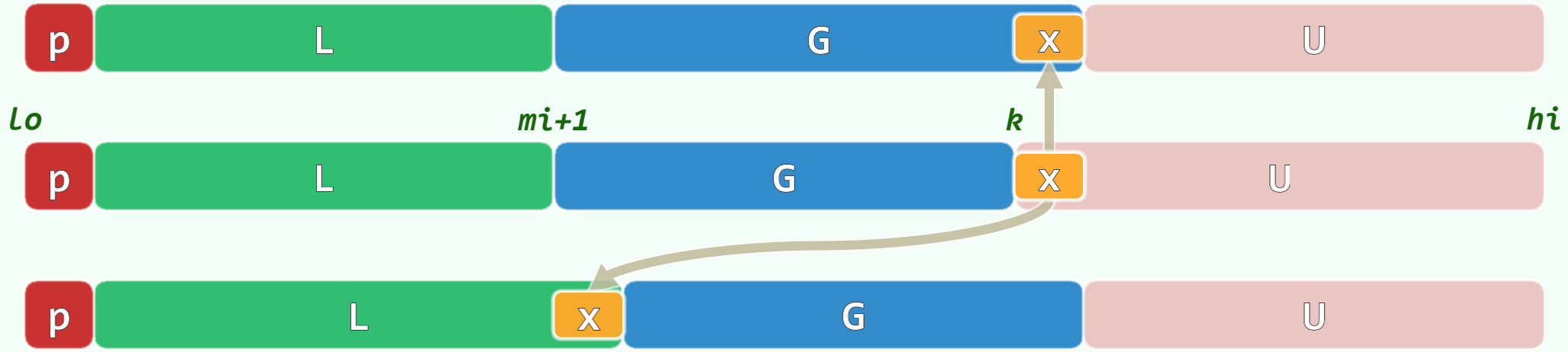
$$L < pivot \leq G$$

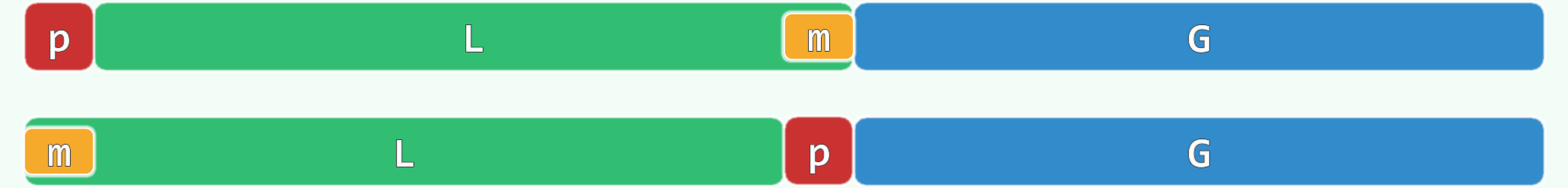




单调性：[k]不小于轴点 ? 直接G拓展 : G滚动后移，L拓展

```
pivot <= S[ k ]  ?  k++  :  swap( S[ ++mi ], S[ k++ ] )
```

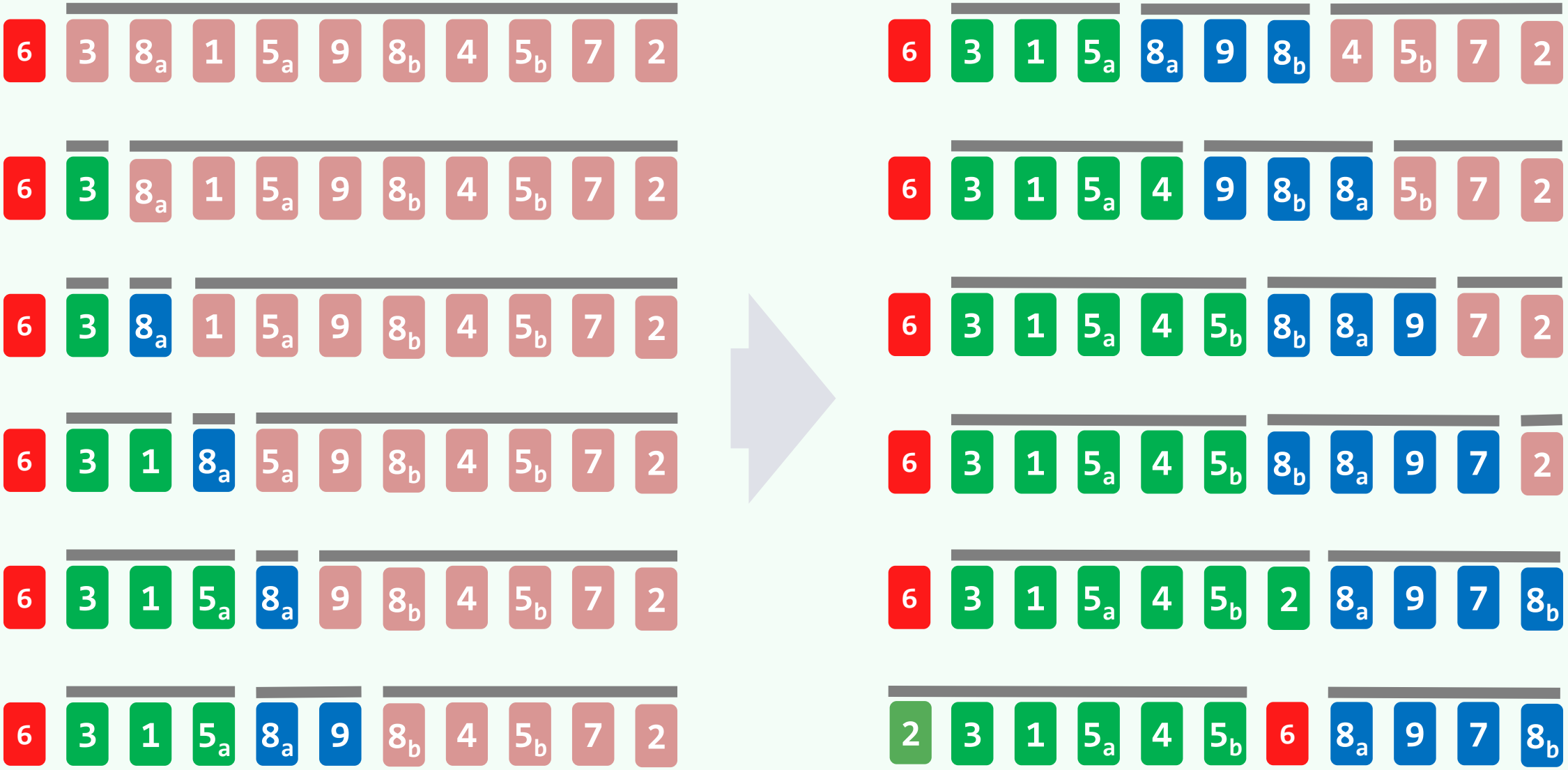




快速划分：LGU版

```
template <typename T> Rank Vector<T>::partition( Rank lo, Rank hi ) { //[lo, hi)
   swap( _elem[ lo ], _elem[ lo + rand() % ( hi - lo ) ] ); //随机轴点
   T pivot = _elem[ lo ]; Rank mi = lo;
   for ( Rank k = lo + 1; k < hi; k++ ) //自左向右考查每个[k]
      if ( _elem[ k ] < pivot ) //若[k]小于轴点，则将其
         swap( _elem[ ++mi ], _elem[ k ]); //与[mi]交换，L向右扩展
   swap( _elem[ lo ], _elem[ mi ] ); //候选轴点归位（从而名副其实）
   return mi; //[lo,mi) < [mi] <= (mi,hi)
}
```

实例


6 3 8a 1 5a 9 8b 4 5b 7 2
6 3 8a 1 5a 9 8b 4 5b 7 2
6 3 8a 1 5a 9 8b 4 5b 7 2
6 3 1 8a 5a 9 8b 4 5b 7 2
6 3 1 5a 8a 9 8b 4 5b 7 2
6 3 1 5a 8a 9 8b 4 5b 7 2
6 3 1 5a 8a 9 8b 4 5b 7 2
6 3 1 5a 8a 9 8b 4 5b 7 2
6 3 1 5a 4 5b 8b 8a 9 7 2
6 3 1 5a 4 5b 8b 8a 9 7 2
6 3 1 5a 4 5b 2 8a 9 7 8b
2 3 1 5a 4 5b 6 8a 9 7 8b

14-B1

排序

选取：众数

善鈞，從眾。夫善，眾之主也。三卿為主，可謂眾矣。從之，不亦可乎？！

诚若为今立计，所当稽求既往，相度方来，掊物质而张灵明，任个人而排众数

然而，在现代文明社会里，居有其所的家庭却不到一半；在文明特别发达的大城市里，拥有住房的人只占全体居民的极小部分

邓俊辉
deng@tsinghua.edu.cn

选取 + 中位数

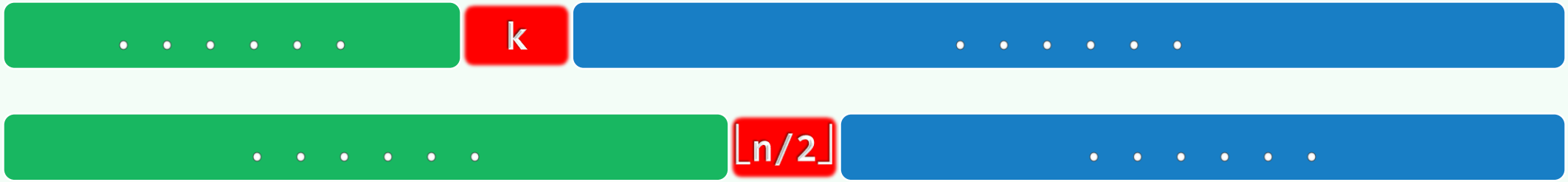
❖ k-selection 在任意一组可比较大小的元素中，如何由小到大，找到次序为 k 者?

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 $S[\lfloor n/2 \rfloor]$ 称作中位数 //与之相等者也是

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

Majority

❖ 无序向量中，若有一半以上元素同为m，则称之为众数

- 在 $\{ \boxed{3}, 5, 2, \boxed{3}, \boxed{3} \}$ 中，众数为3；然而
- 在 $\{ \boxed{3}, 5, 2, \boxed{3}, \boxed{3}, 0 \}$ 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，则亦必是中位数

❖ 于是，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 majCheck( 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 majCheck( A, maj = mode( A ) ); }
```

❖ 同样地，mode()算法难以兼顾时间、空间的高效

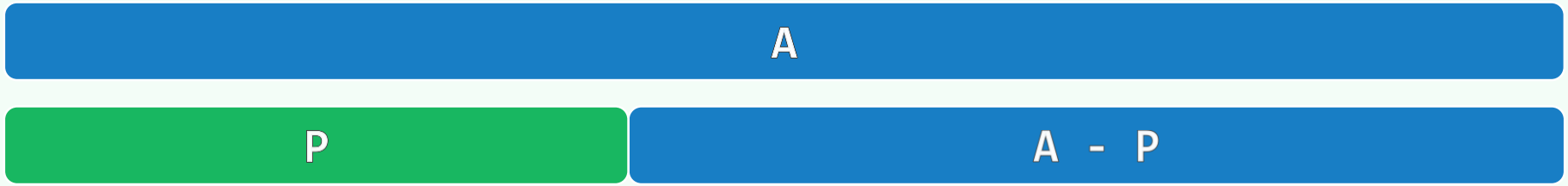
❖ 可行思路：借助功能更弱但计算成本更低的必要条件，选出唯一的候选者

```
template <typename T> bool majority( Vector<T> A, T & maj )
   { return majCheck( A, maj = majCandidate( 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$O(n)$时间做验证，故而只需考虑A的确含有众数的两种情况：

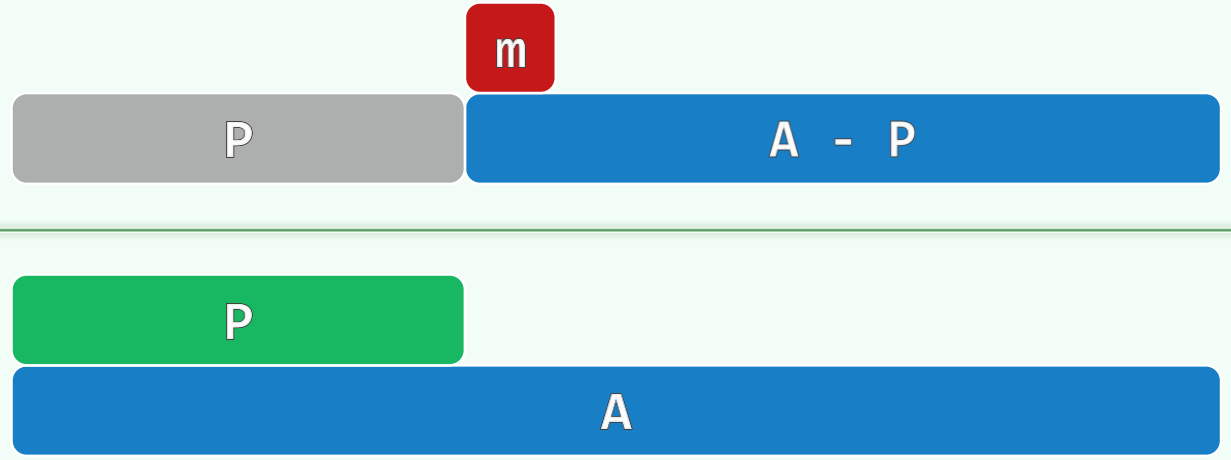
1. 若x = m，则在排除前缀P之后，m与其它元素在数量上的差距保持不变

2. 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不致缩小

❖ 若将众数的标准从“一半以上”改作“至少一半”，算法需做什么调整?

算法

```
template <typename T> T majCandidate( Vector<T> A ) {
   T maj;
   for ( Rank c = 0, i = 0; i < A.size(); i++ )
      if ( 0 == c ) {
         maj = A[i]; c = 1;
      } else
         maj == A[i] ? c++ : c--;
   return maj;
}
```



14-B2

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

德性是两种恶—过度与不及—的中间。在感情与实践中，恶要么达不到正确，要么超过正确。德性则找到并且选取那个正确。所以虽然从其本质或概念来说德性是适度，从最高善的角度来说，它是一个极端

邓俊辉
deng@tsinghua.edu.cn

归并向量的中位数

- 任给有序向量 S_1 和 S_2，长度 n_1 和 n_2

  如何快速找出 $S = S_1 \cup S_2$ 的中位数?

- 蛮力： 经归并得到有序向量 S

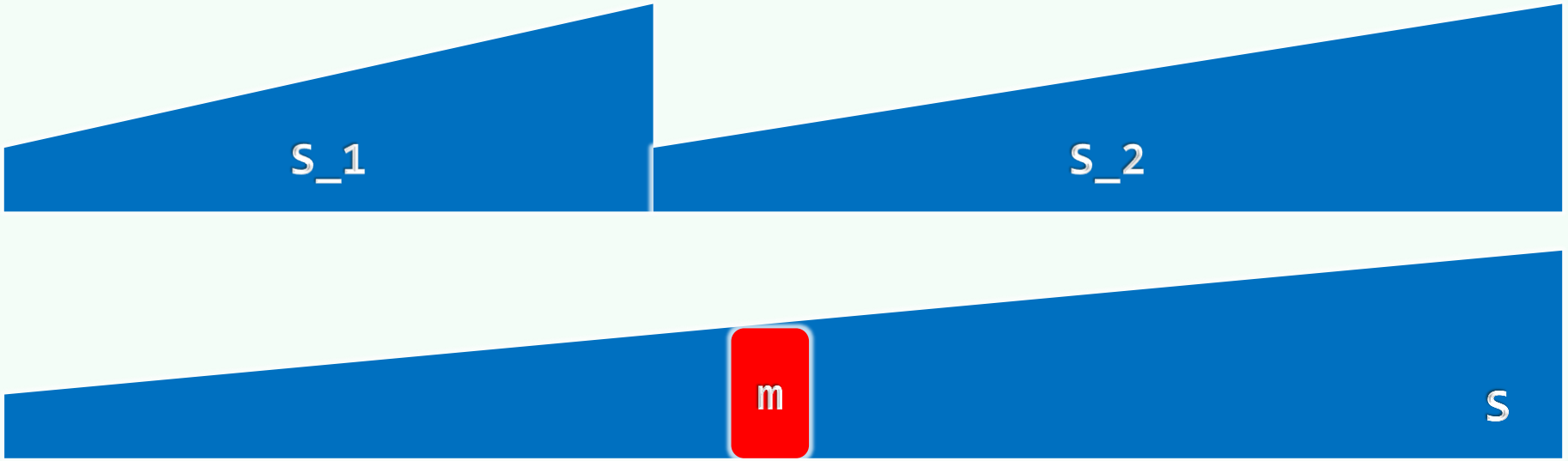
  取 $S[(n_1 + n_2)/2]$ 即是

- 如此，共需 $\mathcal{O}(n_1 + n_2)$ 时间

  但毕竟未能充分利用 S_1 和 S_2 的**有序性**

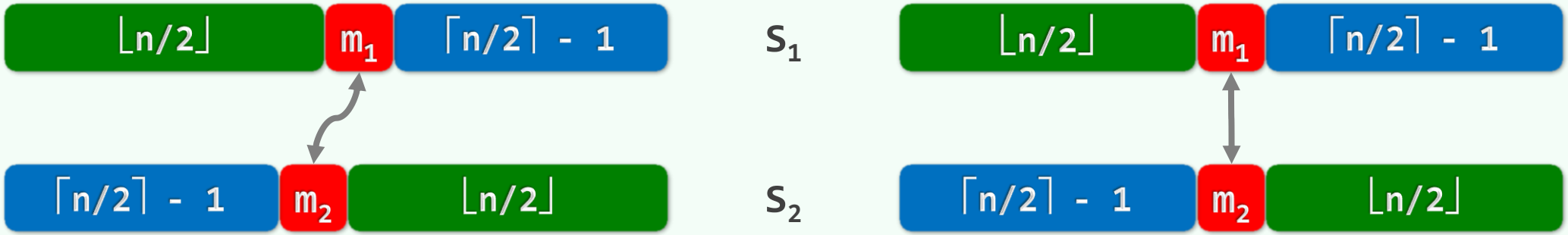
- 以下，先解决 $n_1 = n_2$ 的情况

  依然采用**减而治之**策略...



等长子向量：构思

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



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

❖ 若 $m_1 < m_2$，则 n 无论偶奇，必有：

$$median(S_1 \cup S_2) = median(S_1.suffix(\lceil n/2 \rceil) \cup S_2.prefix(\lceil n/2 \rceil))$$

这意味着，如此减除（一半规模）之后，中位数不变

❖ $m_1 > m_2$ 时同理

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, Rank lo1, Vector<T> & S2, Rank lo2, Rank n ) {
   if ( n < 3 ) return trivialMedian( S1, lo1, n, S2, lo2, n ); //递归基
   Rank 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（1/2）

```
template <typename T>
T median ( Vector<T>& S1, Rank lo1, Rank n1, Vector<T>& S2, Rank lo2, Rank 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（2/2）

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

14-B3

排序

选取：QuickSelect

大胆猜测，小心求证

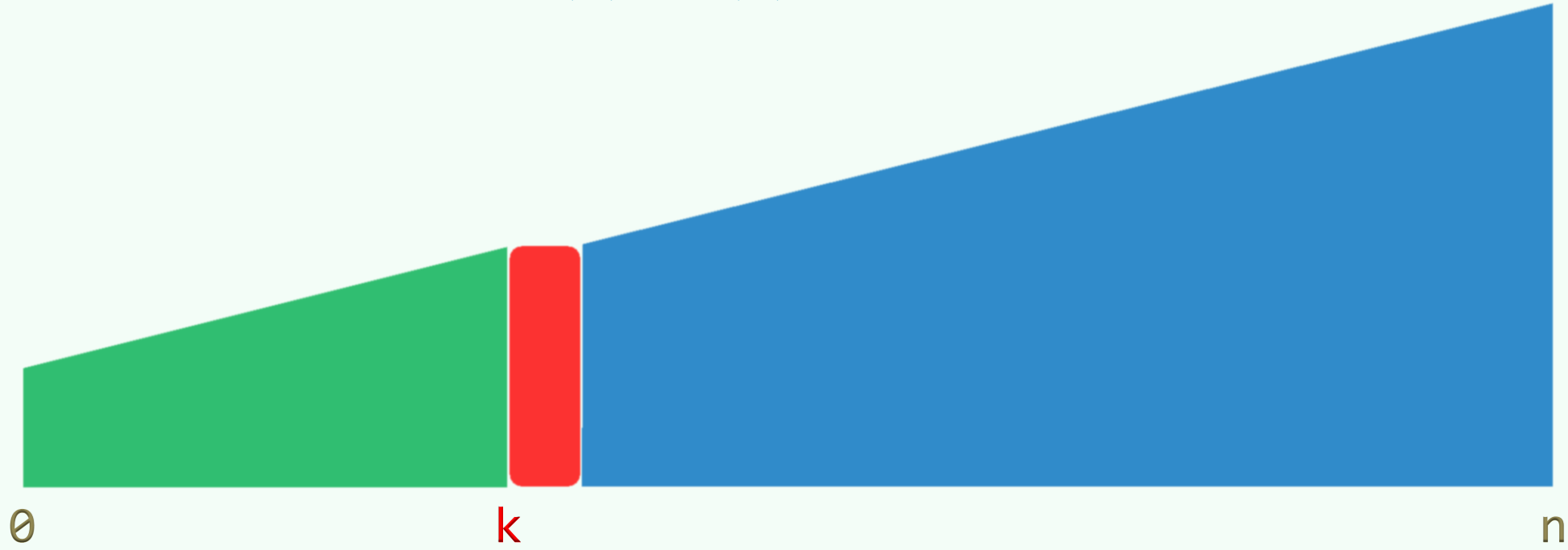
他们在一起谈了一下之后，就转过身来向我表示敬意，对此，我的老师微微一笑；此外，他们还给了我更多的荣誉，因为他们把我列入他们的行列，结果，我就是这样赫赫有名的智者中的第六位

邓俊辉
deng@tsinghua.edu.cn

尝试：蛮力

❖ 对A排序　　//O(nlogn)

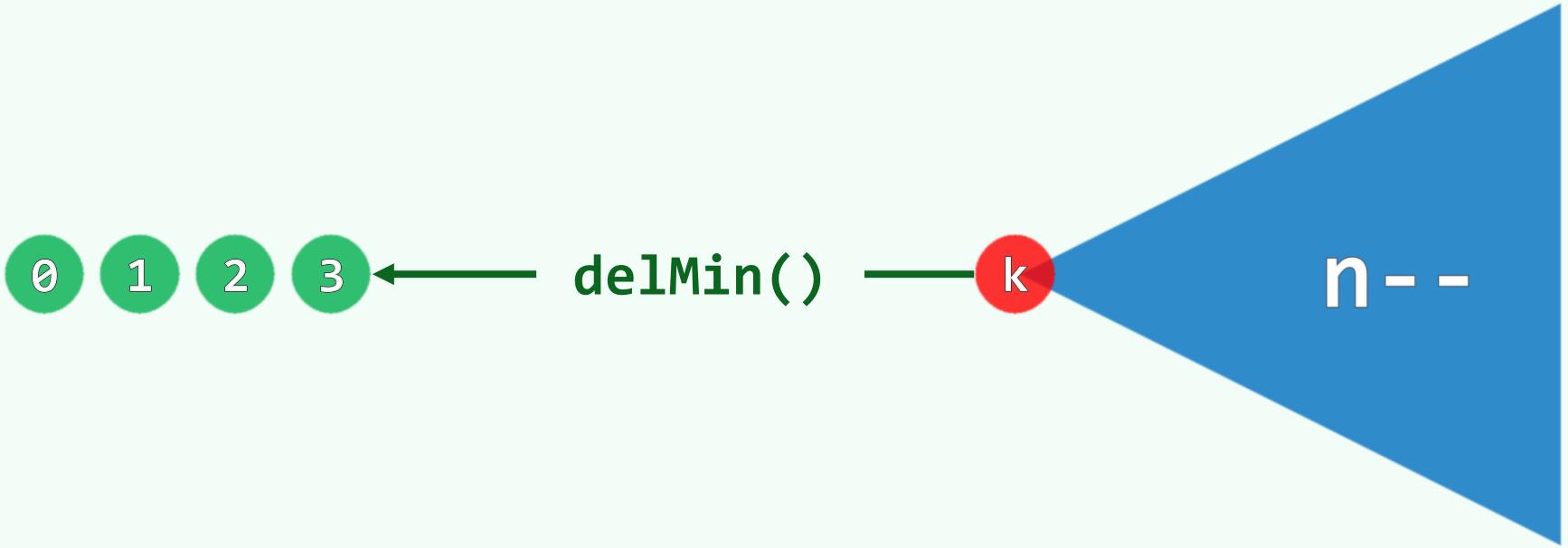
从首元素开始，向后行进k步　　//O(k) = O(n)



尝试：堆（A）

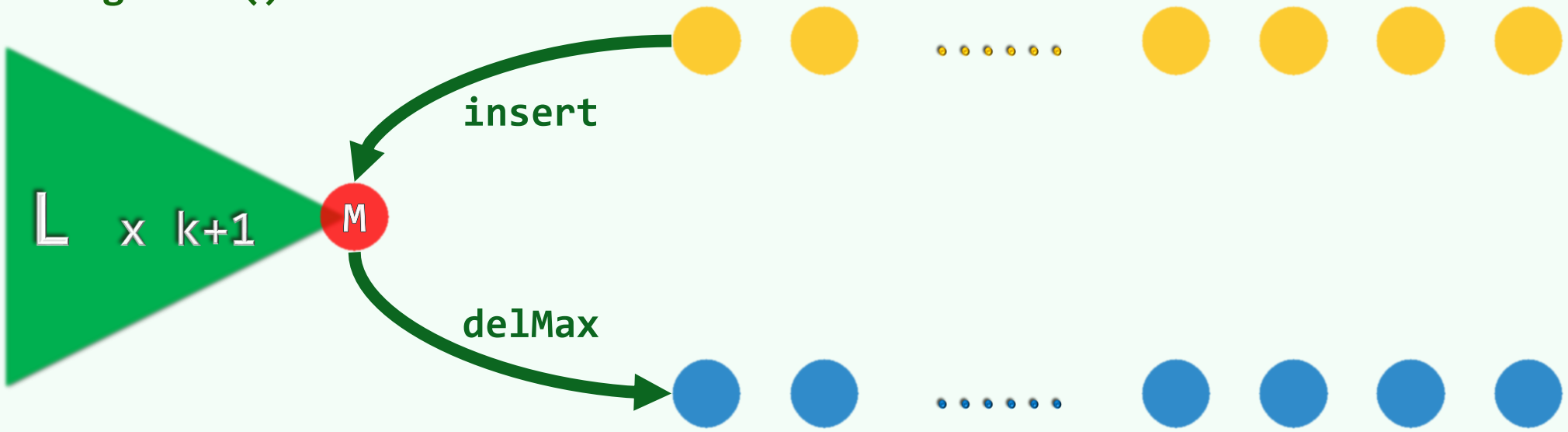
❖ 将所有元素组织为小顶堆　　//O(n)

连续调用k+1次delMin()　　//O(klogn)



尝试：堆（B）

```
❖ L = heapify( A[0, k] ) //任选 k+1 个元素，组织为大顶堆：O(k)
❖ for each i in (k, n) //O(n - k)
     L.insert( A[i] ) //O(logk)
     L.delMax() //O(logk)
  return L.getMax()
```



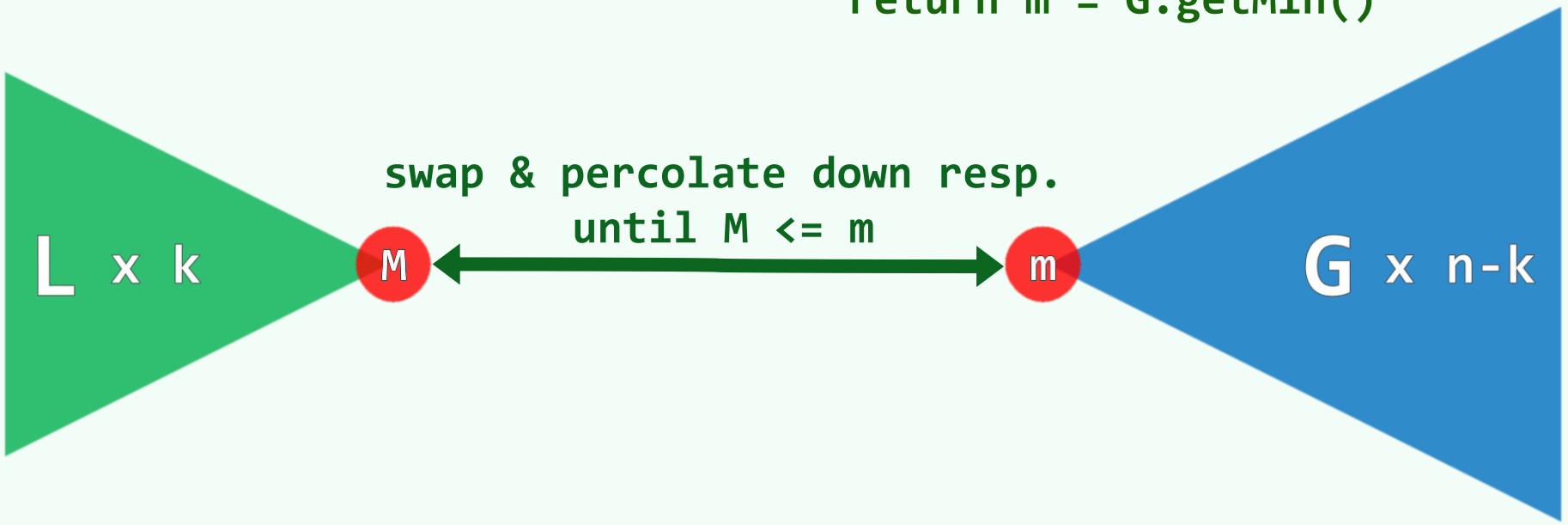
尝试：堆（C）

- 将输入任意划分为规模为k、n-k的子集

  分别组织为大、小顶堆

```
//O(k + (n-k)) = O(n)
```

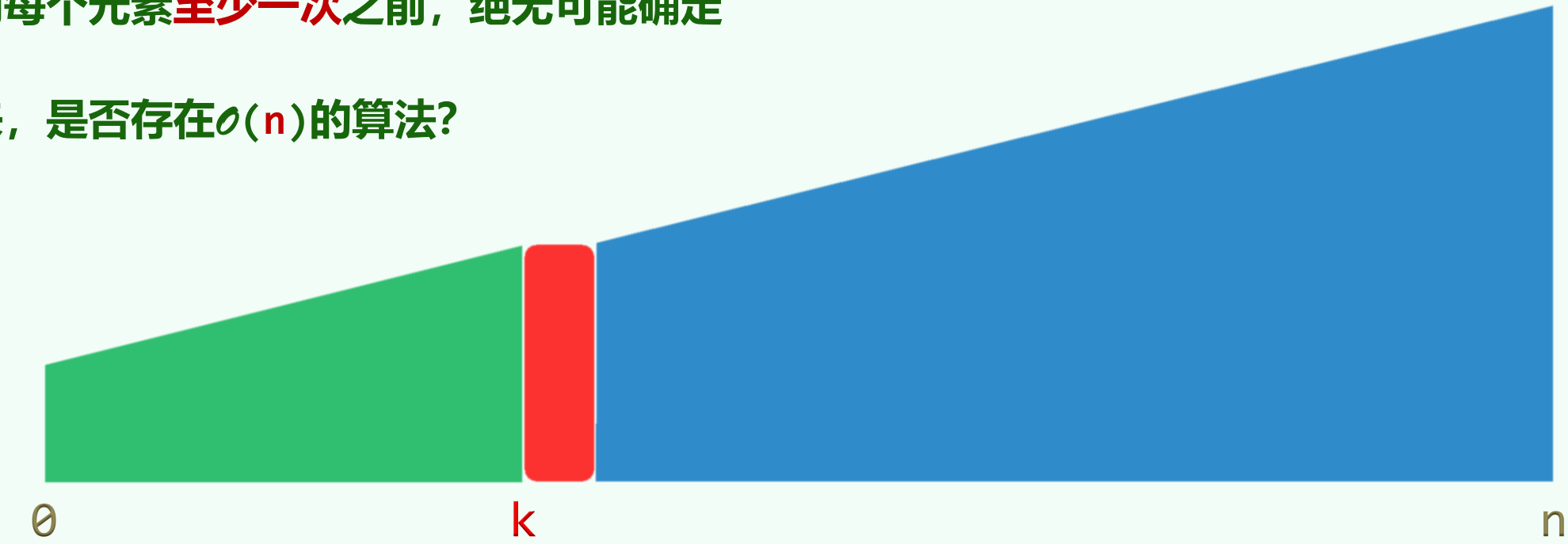
```
while ( M > m ) //O(min(k, n - k))
    swap( M, m )
    L.percolateDown() //O(logk)
    G.percolateDown() //O(log(n - k))
return m = G.getMin()
```



下界与最优

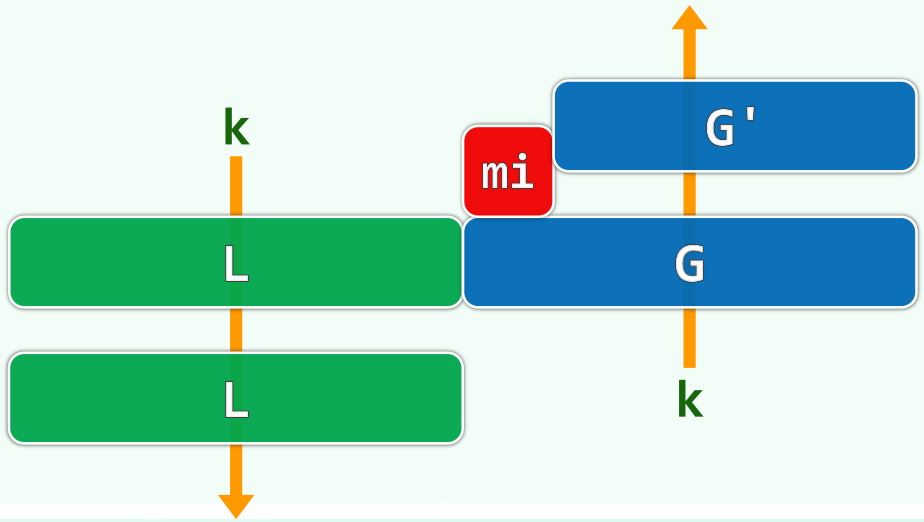
- 是否存在更快的算法？当然，最快也不至于快过Ω(n)！
- 所谓第k小，是相对于序列整体而言，所以...

  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？



快速选取

```
template <typename T> Rank quickSelect( T const * A, Rank n, Rank k ) {
   Vector<Rank> R(n); for ( Rank k = 0; k < n; k++ ) R.insert(k); //使用索引向量，保持原次序
   for ( Rank lo = 0, hi = n; ;  ) { //反复做quickParititon
      swap( R[lo], R[lo + rand()%(hi-lo)] ); T pivot = A[R[lo]]; Rank mi = lo; //大胆猜测
      for ( Rank i = lo+1; i < hi; i++ ) //LGU版partition算法
         if ( A[R[i]] < pivot )
            swap( R[++mi], R[i] );
      swap( R[lo], R[mi] ); //[0,mi) < [mi] <= (mi, n)
      if ( mi < k ) lo = mi + 1; //猜小了，则剪除前缀
      else if ( k < mi ) hi = mi; //猜大了，则剪除后缀
      else return return R[mi]; //或早或迟，总能猜中
   }
}
```



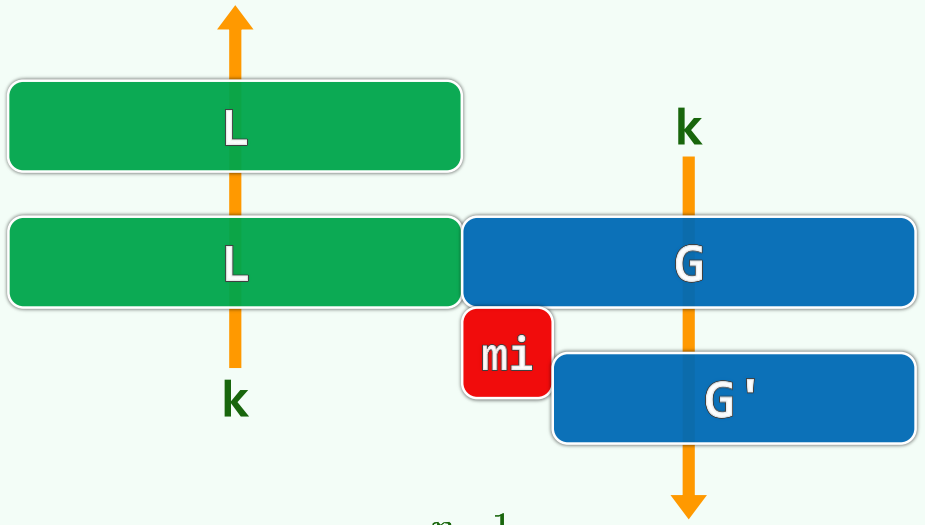
期望性能

❖ 记期望的比较次数为 $T(n)$，于是：

$$T(1) = 0,\ T(2) = 1,\ \ldots$$

$$T(n) = (n-1) + \frac{1}{n} \times \sum_{k=0}^{n-1} \max\{T(k), T(n-k-1)\}$$

$$= (n-1) + \frac{1}{n} \times \sum_{k=0}^{n-1} T(\max\{k, n-k-1\}) \leq (n-1) + \frac{2}{n} \times \sum_{k=n/2}^{n-1} T(k)$$



❖ 事实上，不难验证：$T(n) < 4 \cdot n = T(n) = \mathcal{O}(n)$ ...

$$T(n) \leq (n-1) + \frac{2}{n} \times \sum_{k=n/2}^{n-1} 4k \leq (n-1) + 3n < 4n$$

14-B4

排序

选取：LinearSelect

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

为埃皮奈先生生产的水果，尽管给人偷掉了四分之三，
还比他在舍弗莱特的那片大菜园要多

邓俊辉
deng@tsinghua.edu.cn

linearSelect(A, n, k)

Let Q be a small constant

0. if (n = |A| < Q) return trivialSelect(A, n, k)

1. else divide A evenly into n/Q subsequences (each of size Q)

2. Sort each subsequence and determine n/Q medians //e.g. by insertionsort

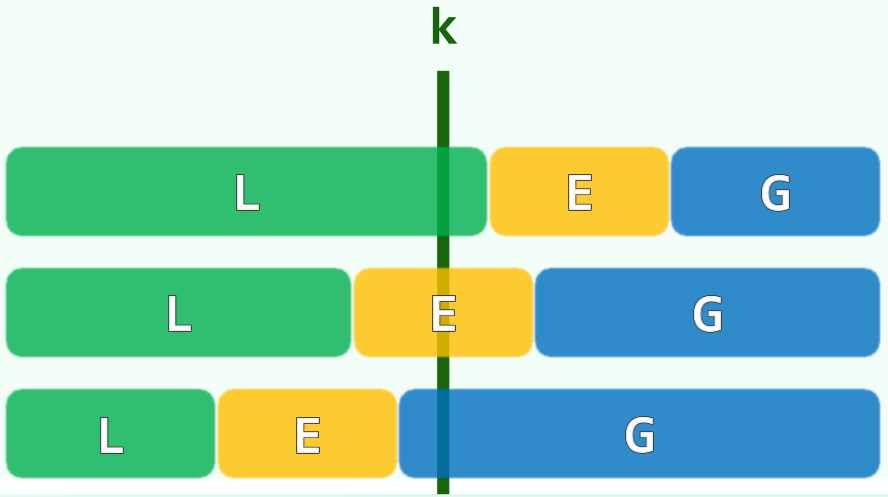
3. Call linearSelect() to find M, median of the medians //by recursion

4. Let L/E/G = { x </=/> M | x ∈ A }

5. if (k < |L|) return linearSelect(A, |L|, k)

   if (k < |L|+|E|) return M

   return linearSelect(A+|L|+|E|, |G|, k-|L|-|E|)



复杂度

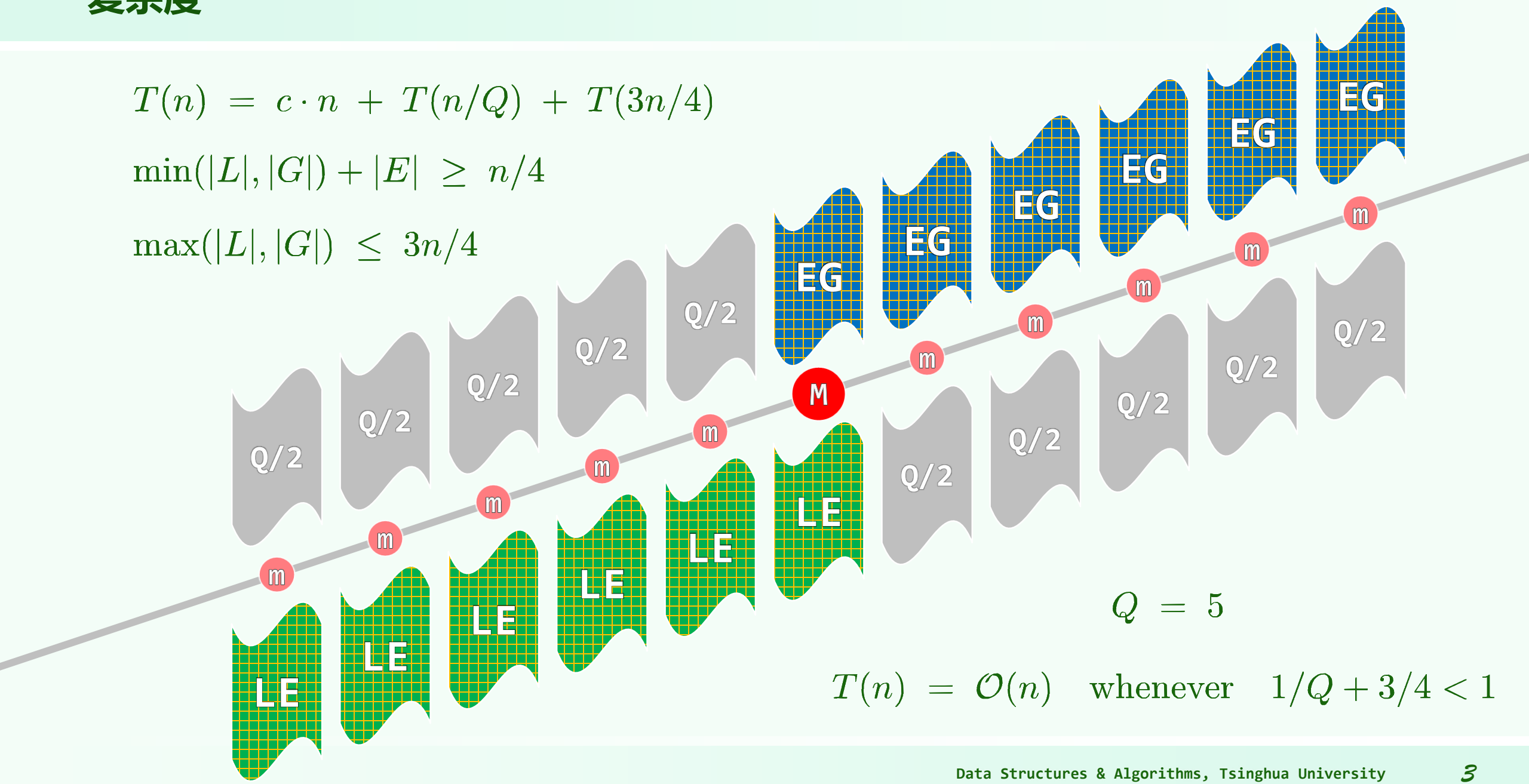
- 将linearSelect()算法的运行时间记作T(n)
- 第0步：$O(1) = O(Q\log Q)$ //递归基：序列长度|A| ≤ Q
- 第1步：$O(n)$ //子序列划分
- 第2步：$O(n) = Q^2 \times n/Q$ //子序列各自排序，并找到中位数
- 第3步：$T(n/Q)$ //从n/Q个中位数中，递归地找到全局中位数
- 第4步：$O(n)$ //划分子集L/E/G，并分别计数 —— 一趟扫描足矣
- 第5步：$T(3n/4)$ //为什么...

复杂度

$$T(n) = c \cdot n + T(n/Q) + T(3n/4)$$

$$\min(|L|, |G|) + |E| \geq n/4$$

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



$$Q = 5$$

$$T(n) = \mathcal{O}(n) \quad \text{whenever} \quad 1/Q + 3/4 < 1$$

14-C1

排序

希尔排序：框架 + 实例

瓜熟蒂落，水到渠成

今有物不知其数，三三数之剩二；五五数之剩三；七七数之剩二。 问物几何

邓俊辉

deng@tsinghua.edu.cn

Shellsort

❖ D. L. Shell：将整个序列视作一个矩阵，逐列各自排序

❖ 递减增量（diminishing increment）

- 由粗到细：重排矩阵，使其更窄，再次逐列排序（h-sorting/h-sorted）
- 逐步求精：如此往复，直至矩阵变成一列（1-sorting/1-sorted）

❖ 步长序列（step sequence）：由各矩阵宽度逆向排列而成的序列

$$\mathcal{H} = \{ h_1 = 1, h_2, h_3, \ldots, h_k, \ldots \}$$

❖ 正确性：最后一次迭代，等同于全排序

1-sorted = ordered

实例：$h_5 = 8$


80 23 19 40 85 1 18 92 71 8 96 46 12


80 23 19 40 85 1 18 92
71 8 96 46 12
71 8 19 40 12 1 18 92
80 23 96 46 85


71 8 19 40 12 1 18 92 80 23 96 46 85

实例：$h_4 = 5$


1 8 19 40 12 71 18 85 80 23 96 46 92


71 8 19 40 12
1 18 92 80 23
96 46 85


1 8 19 40 12
71 18 85 80 23
96 46 92


71 8 19 40 12 1 18 92 80 23 96 46 85

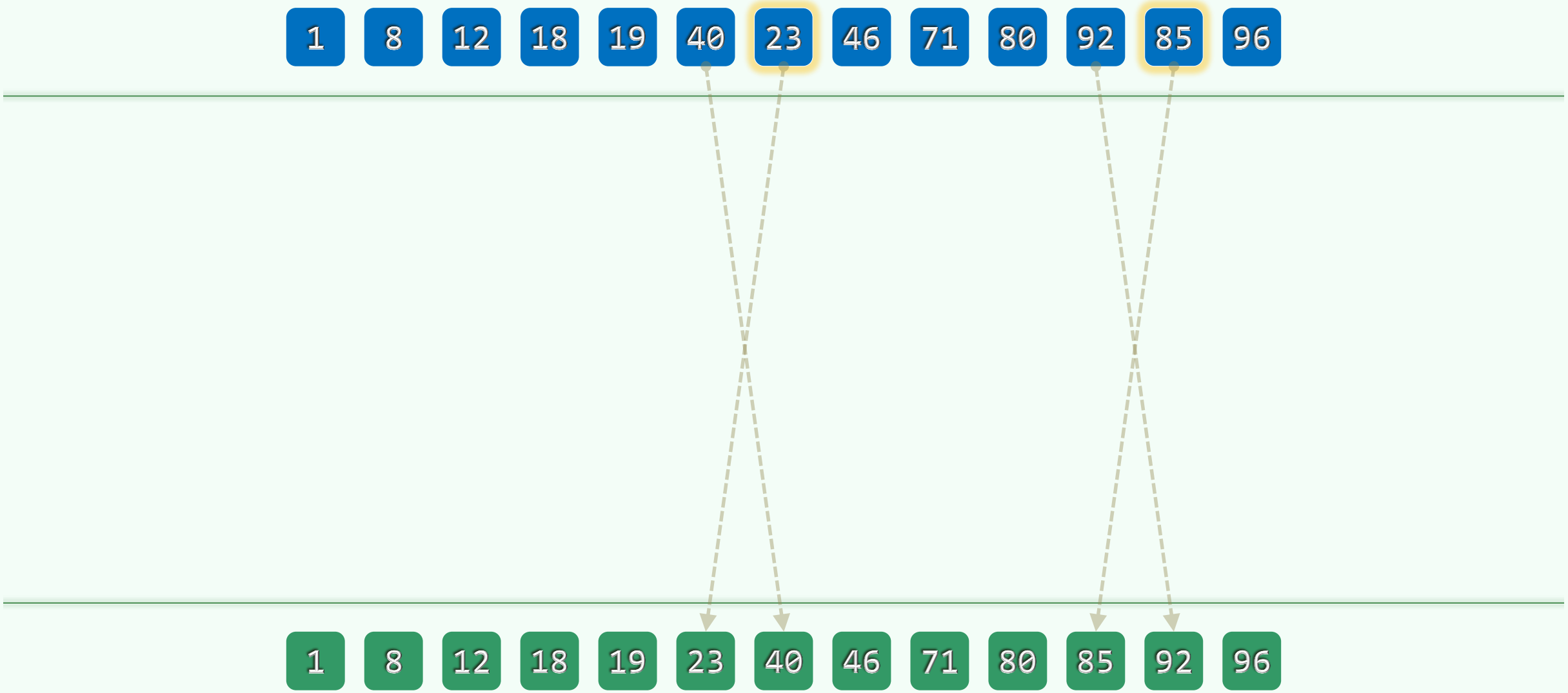
实例：$h_3 = 3$


1 8 19 40 12 71 18 85 80 23 96 46 92
1 40 18 23 92
8 12 85 96
19 71 80 46
1 18 23 40 92
8 12 85 96
19 46 71 80
1 8 19 18 12 46 23 85 71 40 96 80 92

实例：$h_2 = 2$


1 8 12 18 19 40 23 46 71 80 92 85 96
1 19 12 23 71 96 92
8 18 46 85 40 80
1 12 19 23 71 92 96
8 18 40 46 80 85
1 8 19 18 12 46 23 85 71 40 96 80 92

实例：$h_1 = 1$



Call-by-rank

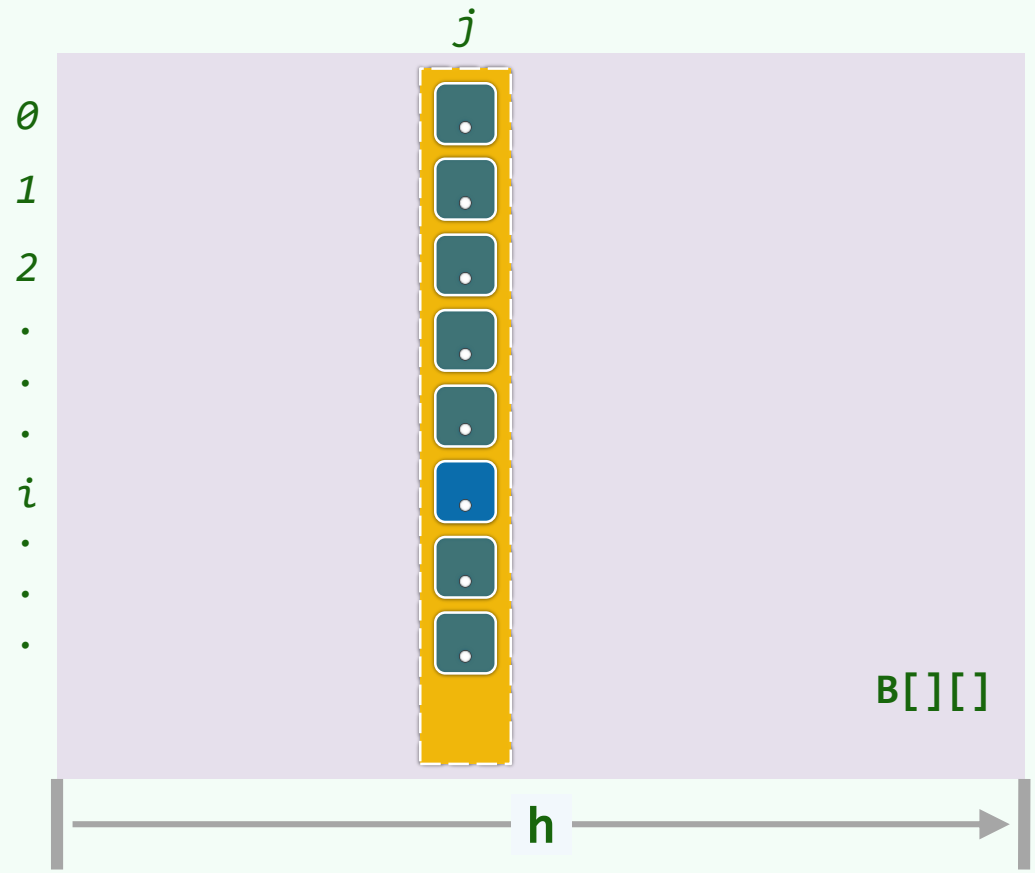
❖ 如何实现矩阵重排？莫非，需要使用二维向量？

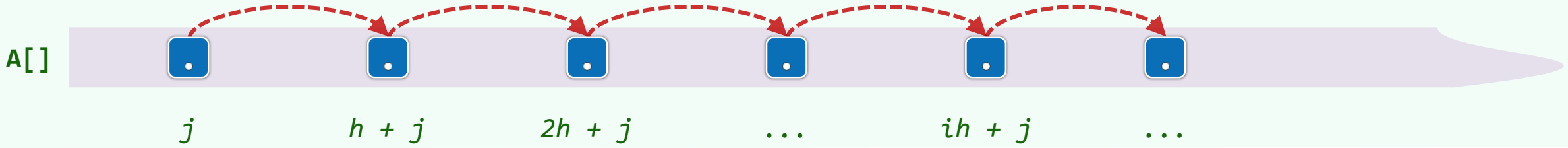
❖ 实际上，借助一维向量足矣

❖ 在每步迭代中，若当前的矩阵宽度为h，则

$$B[i][j] = A[i \cdot h + j]$$

或 $$A[k] = B[k/h][k\%h]$$





实现

```
template <typename T> void Vector<T>::shellSort( Rank lo, Rank hi ) {
   for ( Rank d = 0x7FFFFFFF; 0 < d; d >>= 1 ) //PS Sequence: 1, 3, 7, 15, 31, ...
      for ( Rank j = lo + d; j < hi; j++ ) { //for each j in [lo+d, hi)
         T x = _elem[j]; Rank i = j; //within the prefix of the subsequence of [j]
         while ( (lo + d <= i) && (x < _elem[i-d]) ) //find the appropriate
            _elem[i] = _elem[i-d], i -= d; //predecessor [i]
         _elem[i] = x; //where to insert [j]
      }
} //0 <= lo < hi <= size <= 2^31
```

排序

希尔排序：Shell序列 + 输入敏感性

14-C2

邓俊辉

deng@tsinghua.edu.cn

让上帝的归上帝，凯撒的归凯撒

Shell's Sequence, 1959: $\mathcal{H}_{shell} = \{1, 2, 4, 8, 16, 32, 64, \ldots, 2^k, \ldots\}$

- 实际上，采用 $\mathcal{H}_{shell}$，在最坏情况下需要运行 $\Omega(n^2)$ 时间...
- 考查由子序列 A = unsort[0, 2^{N-1}) 和 B = unsort[2^{N-1}, 2^N) 交错而成的序列

- 在做2-sorting时，A、B各成一列；故此后必然各自有序

- 然而其中的逆序对依然很多，最后的1-sorting仍需 $1+2+3+\cdots+2^{N-1} = \Omega(n^2/4)$ 时间
- 问题的根源在于，$\mathcal{H}_{shell}$中各项并不互素，甚至相邻项也非互素

Postage Problem

- The postage for a letter is 50F, and a postcard 35F

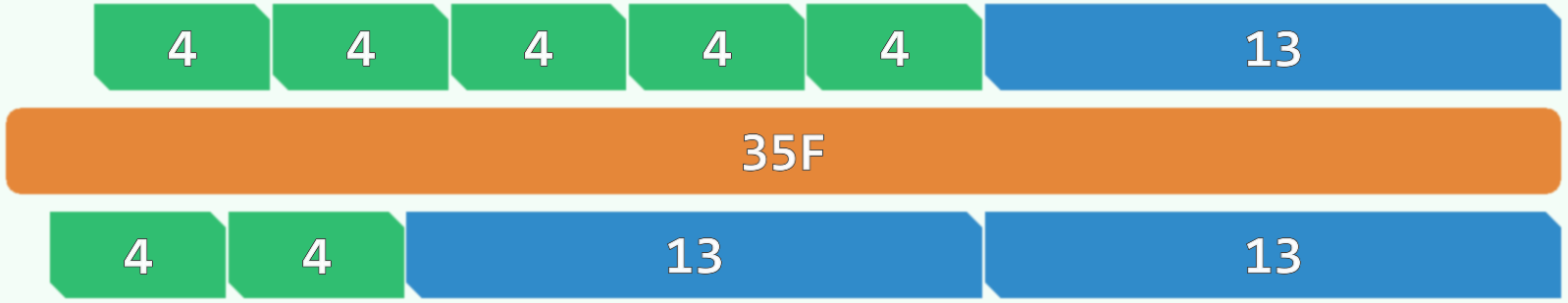
  But there are only stamps of 4F and 13F available

- Possible to stamp

  the letter and

  the postcard

  EXACTLY?



- Given a postage P, determine whether $P \in \{ n \cdot 4 + m \cdot 13 \mid n, m \in \mathcal{N} \}$

Linear Combination

❖ Let $g, h \in \mathcal{N}$

For any $n, m \in \mathcal{N}$, $n \cdot g + m \cdot h$ is called a linear combination of g and h

❖ Denote $\mathbf{C}(g,h) = \{ ng + mh \mid n, m \in \mathcal{N} \}$

$\mathbf{N}(g,h) = \mathcal{N} \backslash \mathbf{C}(g,h)$ //numbers that are NOT combinations of g and h

$\mathbf{x}(g,h) = \max\{ \mathbf{N}(g,h) \}$ //always exists?

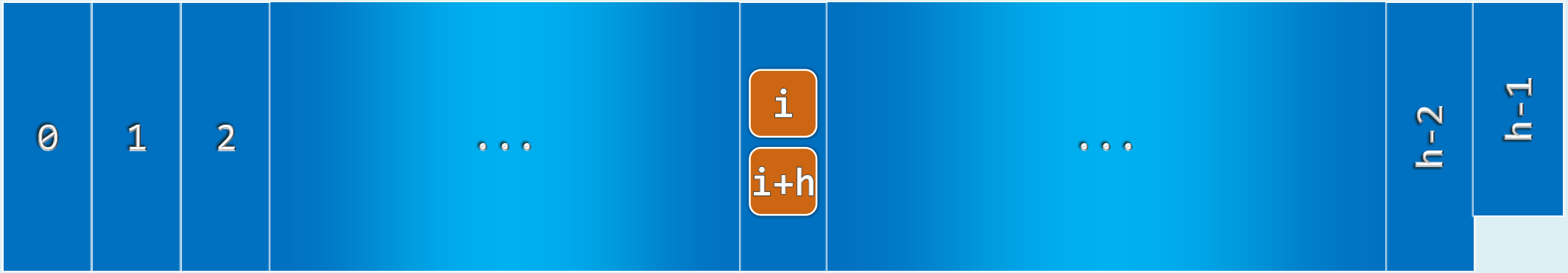
❖ Theorem: when g and h are relatively prime, we have

$$\mathbf{x}(g,h) = (g-1)\cdot(h-1) - 1 = gh - g - h$$

e.g. $\mathbf{x}(3,7) = 11$, $\mathbf{x}(4,9) = 23$, $\mathbf{x}(\boxed{4}, \boxed{13}) = \boxed{35}$, $\mathbf{x}(5,14) = 51$

h-sorting & h-ordered

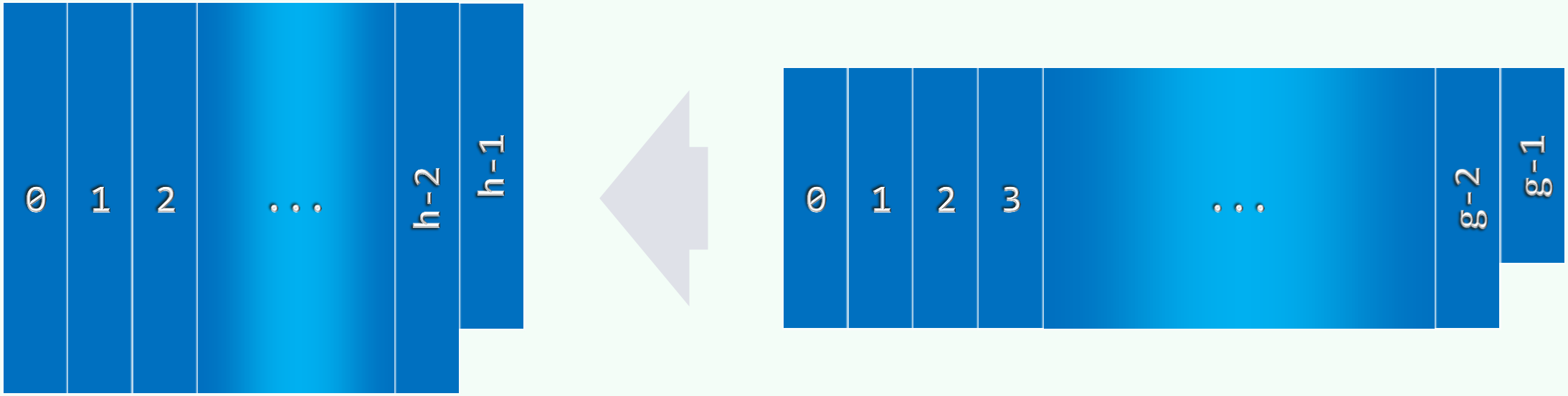
- A sequence S[0,n) is called h-ordered if $S[i] \leq S[i+h],\ \forall\ 0 \leq i < n-h$
- A 1-ordered sequence is sorted
- h-sorting: an h-ordered sequence is obtained by
  - arranging S into a 2D matrix with h columns and
  - sorting each column respectively



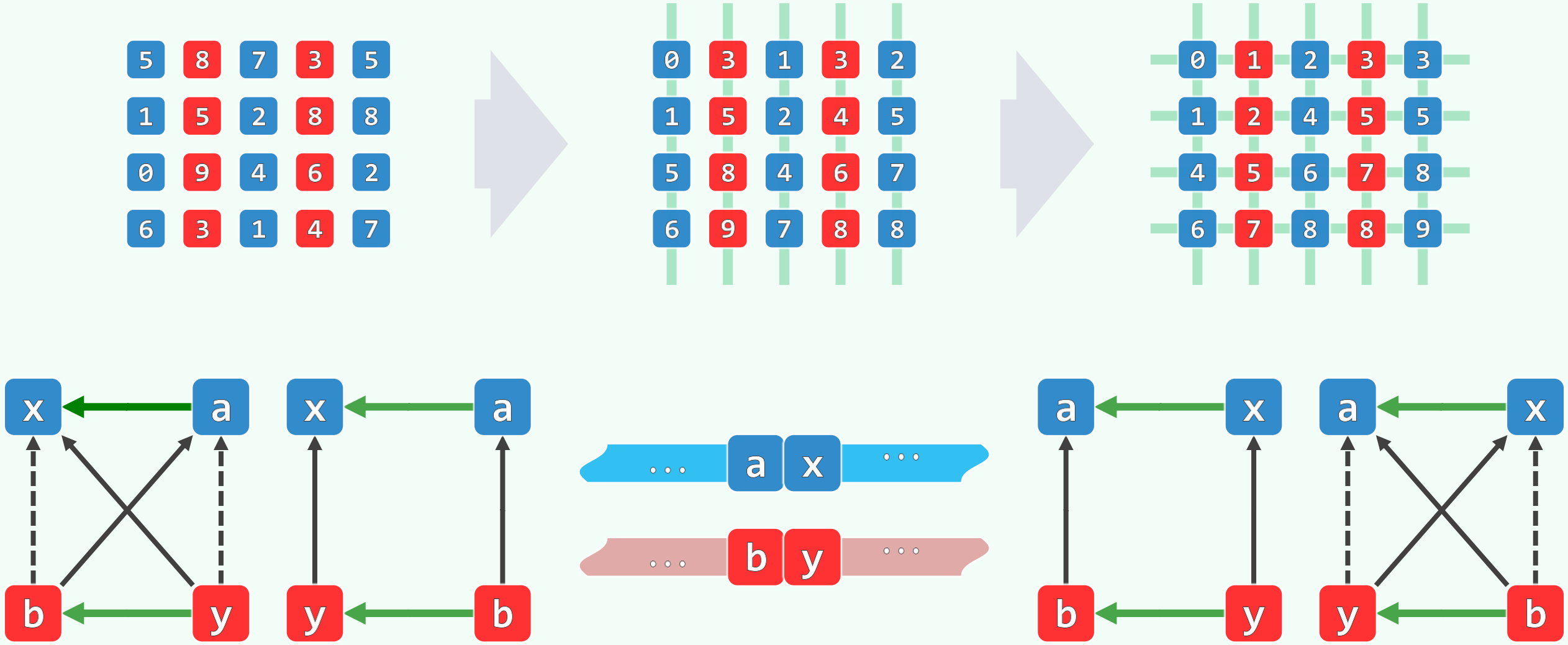
Theorem K

❖ [Knuth, ACP Vol.3 p.90] //习题解析[12-12, 12-13]

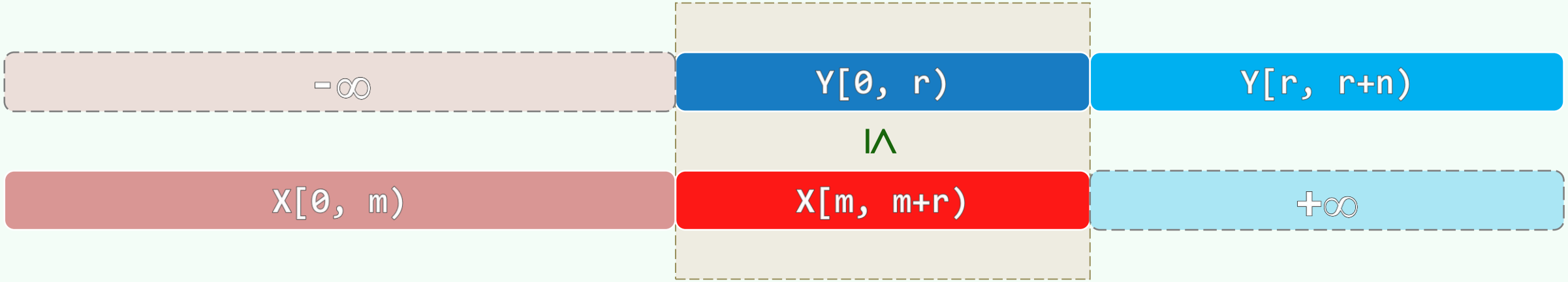
A g-ordered sequence REMAINS g-ordered after being h-sorted.

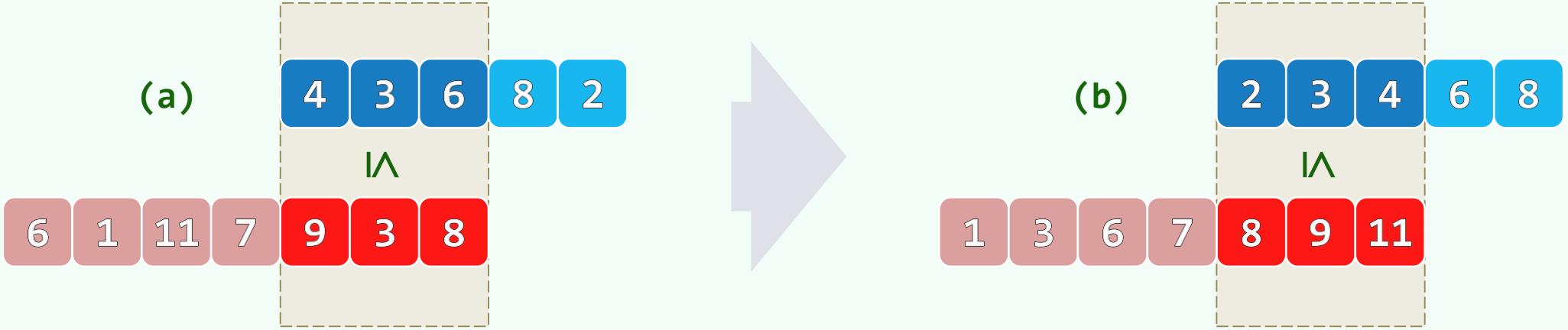


Order Preservation



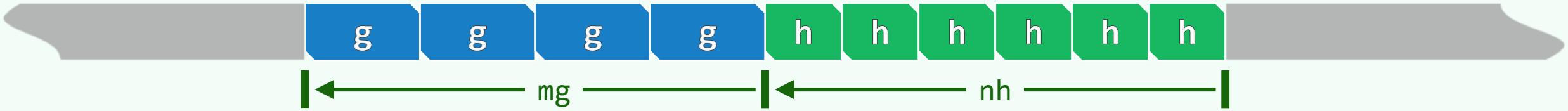
Lemma L

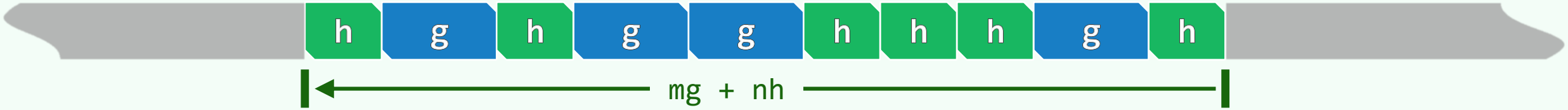




Linear Combination

A sequence that is both g-ordered and h-ordered is called (g,h)-ordered, which must be both (g+h)-ordered and (mg+nh)-ordered for any m, n ∈ N





Inversion

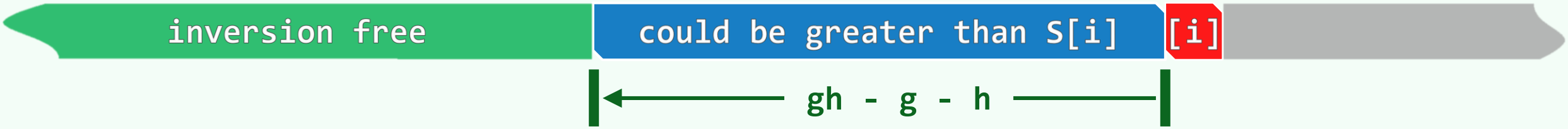
- Let S[0,n) be a (g,h)-ordered sequence, where g and h are relatively prime

  Then for all elements S[j] and S[i], we have

$$i - j > \mathbf{x}(g,h) \qquad \text{only if} \qquad S[j] \leq S[i]$$

- This implies that to the LEFT of each element,

  only the previous $\mathbf{x}(g,h)$ elements could be GREATER



There would be no more than $n \cdot \mathbf{x}(g,h)$ INVERSIONs altogether

14-C3

排序

希尔排序：PS序列

They are like the leaves which a tempest whirls up and scatters in every direction and then allows to fall.

邓俊辉

deng@tsinghua.edu.cn

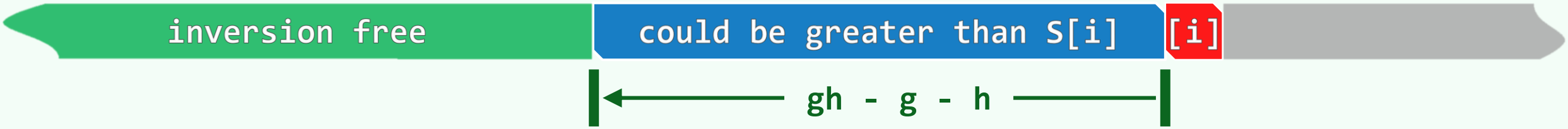
d-Sorting an $\mathcal{O}$(d)-Ordered Sequence in $\mathcal{O}$(dn) Time

❖ If g and h are relatively prime and are both in $\mathcal{O}(d)$

we can d-sort the sequence in $\mathcal{O}(dn)$ time ...

- re-arrange the sequence as a 2D matrix with d columns

- each element is swapped with $\mathcal{O}((g-1)\cdot(h-1)/d) = \mathcal{O}(d)$ elements



❖ Since this holds for all elements, $\mathcal{O}(dn)$ steps are enough

PS Sequence

❖ **Papernov & Stasevic, 1965** //also called Hibbard's sequence

$$\mathcal{H}_{PS} = \mathcal{H}_{Shell} - 1 = \{ 2^k - 1 \mid k \in \mathcal{N} \} = \{ 1, 3, 7, 15, 31, 63, 127, 255, \ldots \}$$

❖ **Different items MAY NOT be relatively prime, e.g.,** $h_{2k} = h_k \cdot (h_k + 2)$

But ADJACENT items MUST be, since $h_{k+1} - 2 \cdot h_k \equiv 1$

❖ **Shellsort with** $\mathcal{H}_{PS}$ **needs**

- $\mathcal{O}(\log n)$ **outer iterations and**
- $\mathcal{O}(n^{3/2})$ **time to sort a sequence of length n** //Why ...

t < k

❖ Let h_t be the h closest to $\sqrt{n}$ and hence $h_t \approx \sqrt{n} = \Theta(n^{1/2})$

1) Consider those iterations for $\{ h_k \mid t < k \} = \{ \overleftarrow{h_{t+1}, h_{t+2}, \ldots, h_m} \}$

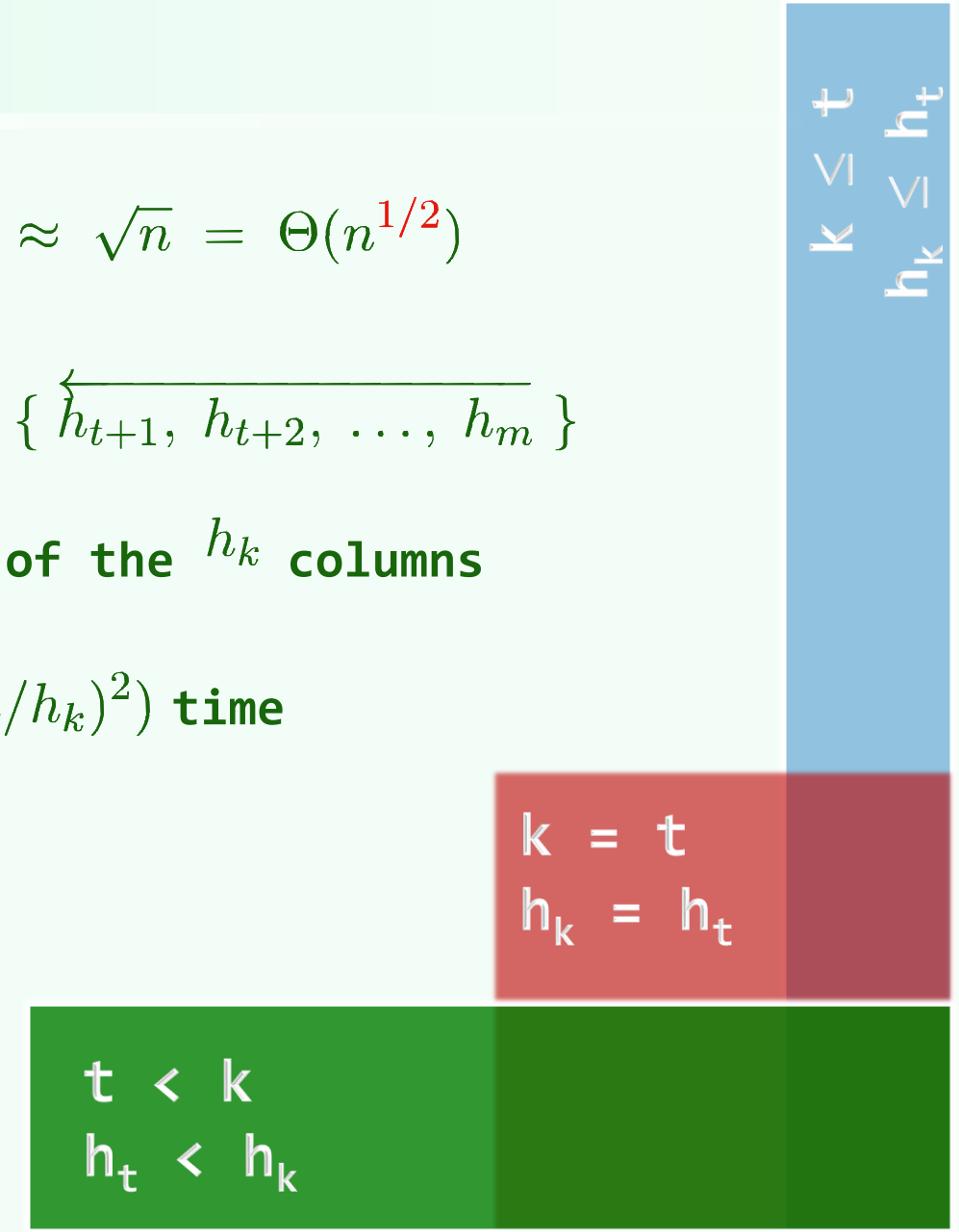
∵ there would be $\mathcal{O}(n/h_k)$ elements in each of the h_k columns

∴ we can **insertionsort** each column in $\mathcal{O}((n/h_k)^2)$ time

∴ each h_k-sorting costs $\mathcal{O}(n^2/h_k)$ time

∴ all these iterations cost time of

$$\mathcal{O}(2 \times n^2/h_t) = \mathcal{O}(n^{3/2})$$



k ≤ t

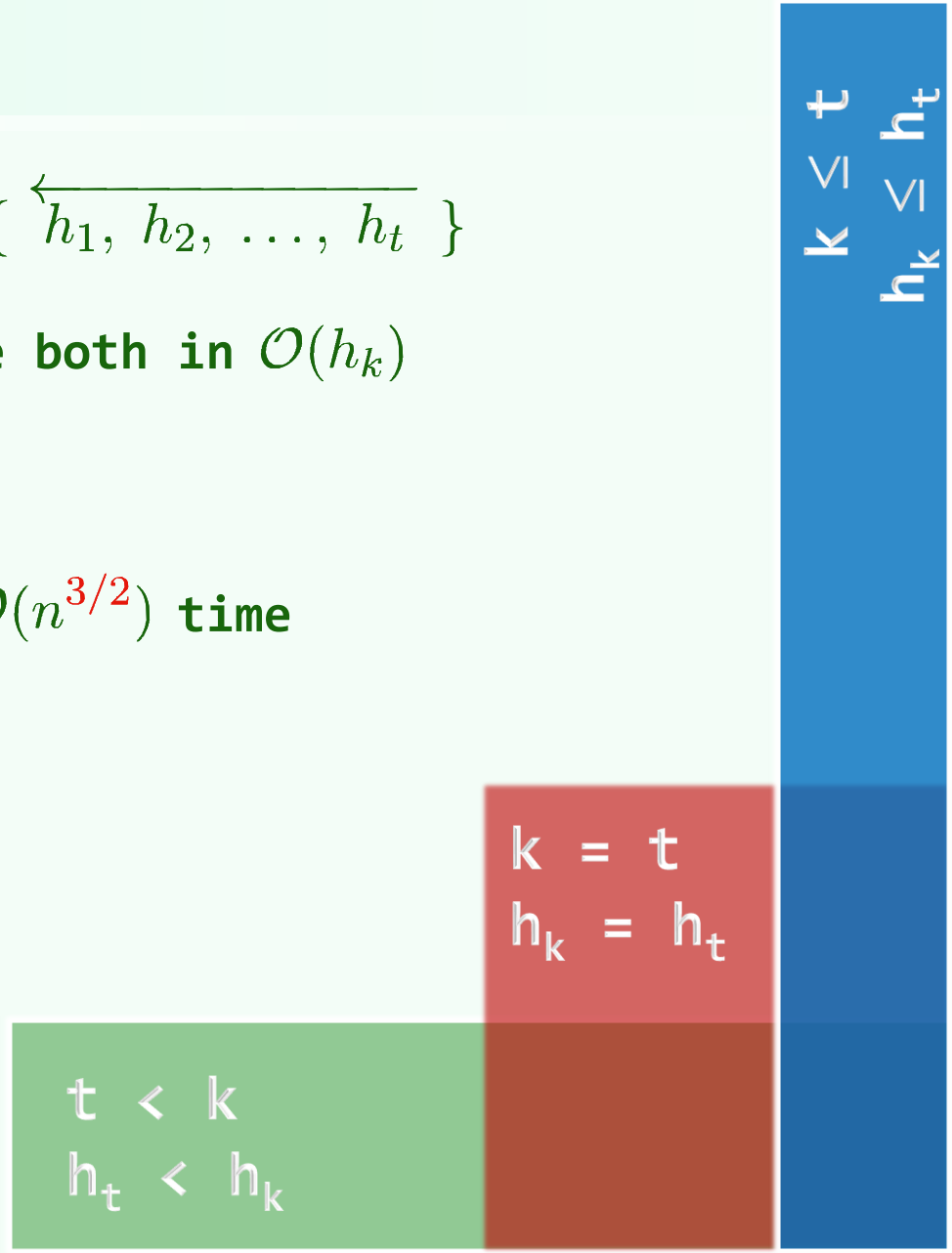
2) Consider those iterations for $\{ h_k \mid k \le t \} = \{ \overleftarrow{h_1, h_2, \ldots, h_t} \}$

$\because$ h_{k+1} and h_{k+2} are relatively prime and are both in $\mathcal{O}(h_k)$

$\therefore$ each h_k-sorting costs $\mathcal{O}(n \times h_k)$ time

$\therefore$ all these iterations cost $\mathcal{O}(n \times 2 \cdot h_t) = \mathcal{O}(n^{3/2})$ time

- This upper bound is TIGHT
- What about the average cases?
  - $\mathcal{O}(n^{5/4})$ based on simulations
  - but not proved yet



14-C4

排序

希尔排序：Pratt序列

聚沙成塔，集腋成裘

邓俊辉
deng@tsinghua.edu.cn

Pratt's Sequence, 1971

$$\mathcal{H}_{pratt} = \{ 2^p \cdot 3^q \mid p, q \in \mathcal{N} \}$$

$$= \{ 1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, \ldots \}$$

❖ Note that

- adjacent items are NOT always relatively prime and
- there are $\mathcal{O}(\log^2 n)$ items no greater than n

❖ With $\mathcal{H}_{pratt}$,

Shellsort sorts a sequence of length n in $\mathcal{O}(n \cdot \log^2 n)$ time ...

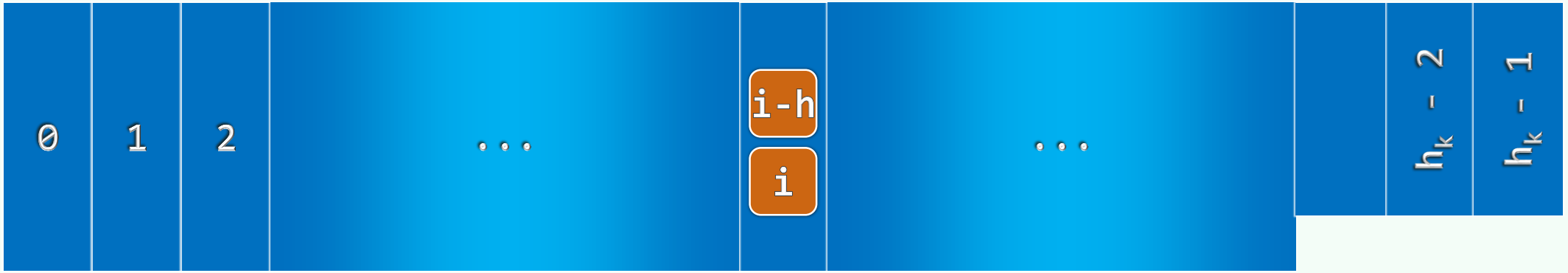
From (2,3)-ordered to 1-ordered

$\because \ \mathbf{x}(2,3) = 2 \cdot 3 - 2 - 3 = 1$

$\therefore$ To the **LEFT** of each element in a (2,3)-ordered sequence,

only the **NEXT** element can be smaller

$\therefore$ It costs $\mathcal{O}(n)$ time to sort such a sequence



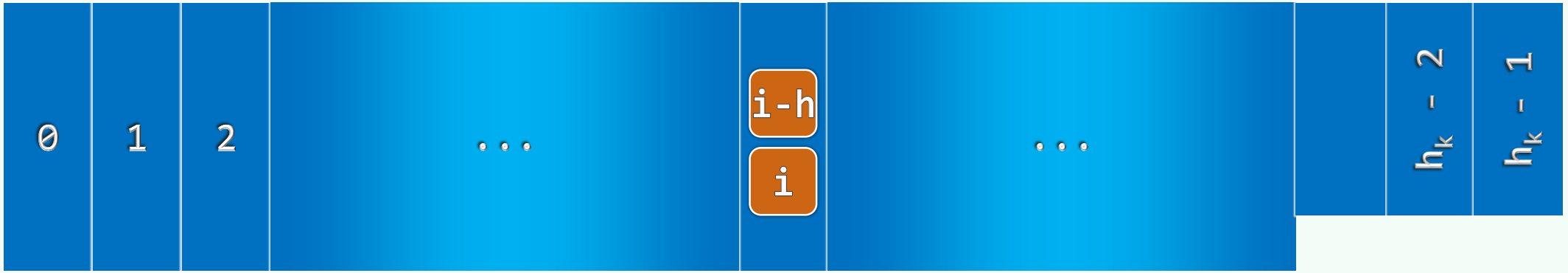
From $(2*h_k, 3*h_k)$-ordered to h_k-ordered

- Divide S into h_k subsequences, each of which is (2,3)-ordered

  $\therefore$ it costs altogether $\mathcal{O}(n)$ time to sort them resp.

- $\because$ there are altogether $\mathcal{O}(\log^2 n)$ iterations

  $\therefore$ we need $\mathcal{O}(n \cdot \log^2 n)$ time



14-C5

排序

希尔排序：Sedgewick序列

邓俊辉

deng@tsinghua.edu.cn

Sedgewick's Sequence

❖ Pratt's sequence requires too many iterations and hence is not good for pre-sorted sequences

❖ Sedgewick's sequence: a combination of PS's sequence with Pratt's

$$\{\ 1,\ 5,\ 19,\ 41,\ 109,\ 209,\ 505,\ 929,\ 2161,\ 3905,\ 8929,\ 16001,\ 36289,\ 64769,\ \cdots\ \}$$

$$\{\ 9\times 4^k - 9\times 2^k + 1 \mid k \geq 0\ \} \quad \cup \quad \{\ 4^k - 3\times 2^k + 1 \mid k \geq 2\ \}$$

- worst $\mathcal{O}(n^{4/3})$ & average $\mathcal{O}(n^{7/6})$
- best performance in practice

❖ Is there a step sequence with $\mathcal{O}(n \cdot \log n)$ worst-case performance?