# 14-A3

## 排序

### 快速排序：迭代、贪心与随机

察一叶而知天下秋

瑕不掩瑜，瑜不掩瑕，忠也

邓俊辉

deng@tsinghua.edu.cn

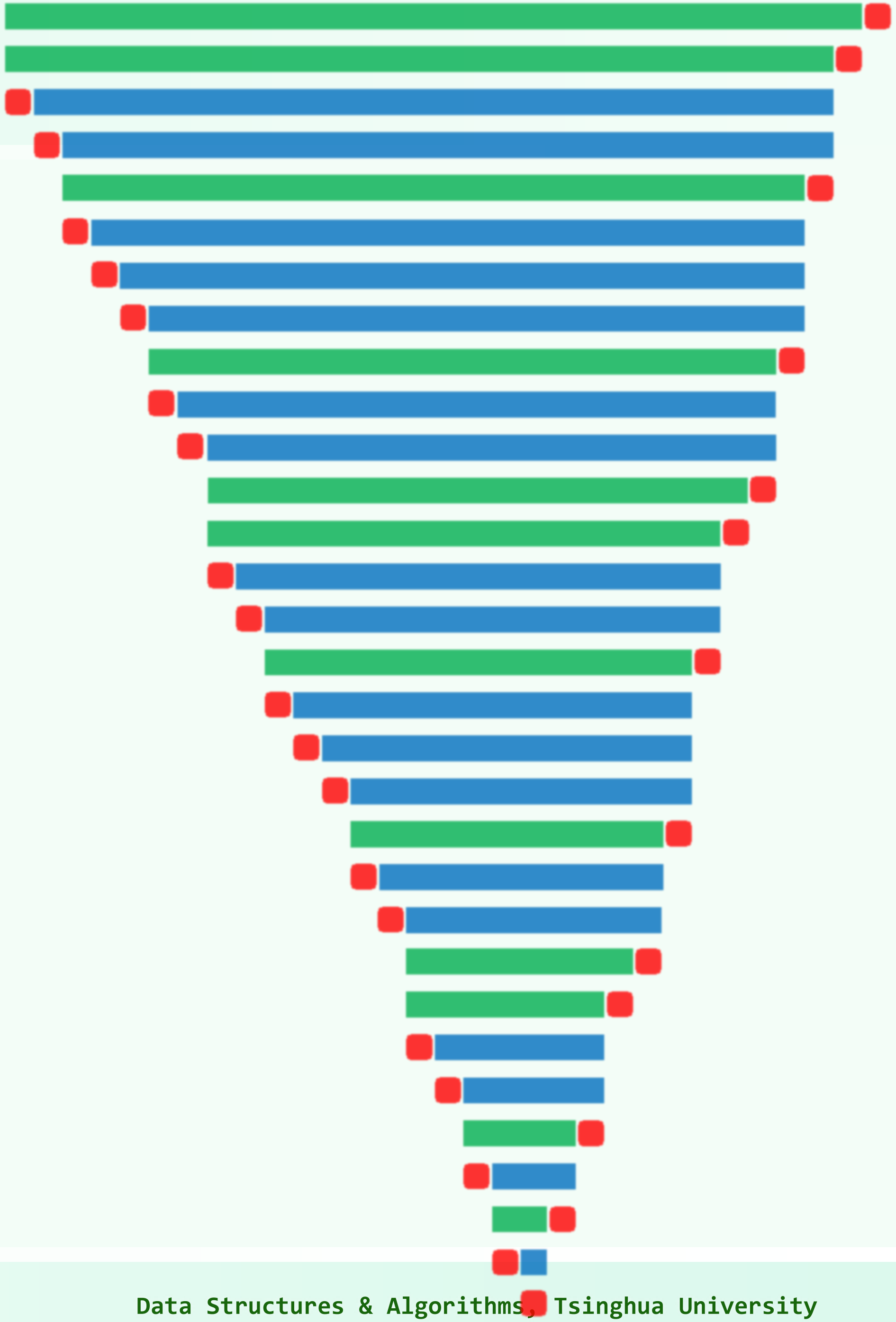
# 空间复杂度 ~ 递归深度

- ❖ 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？

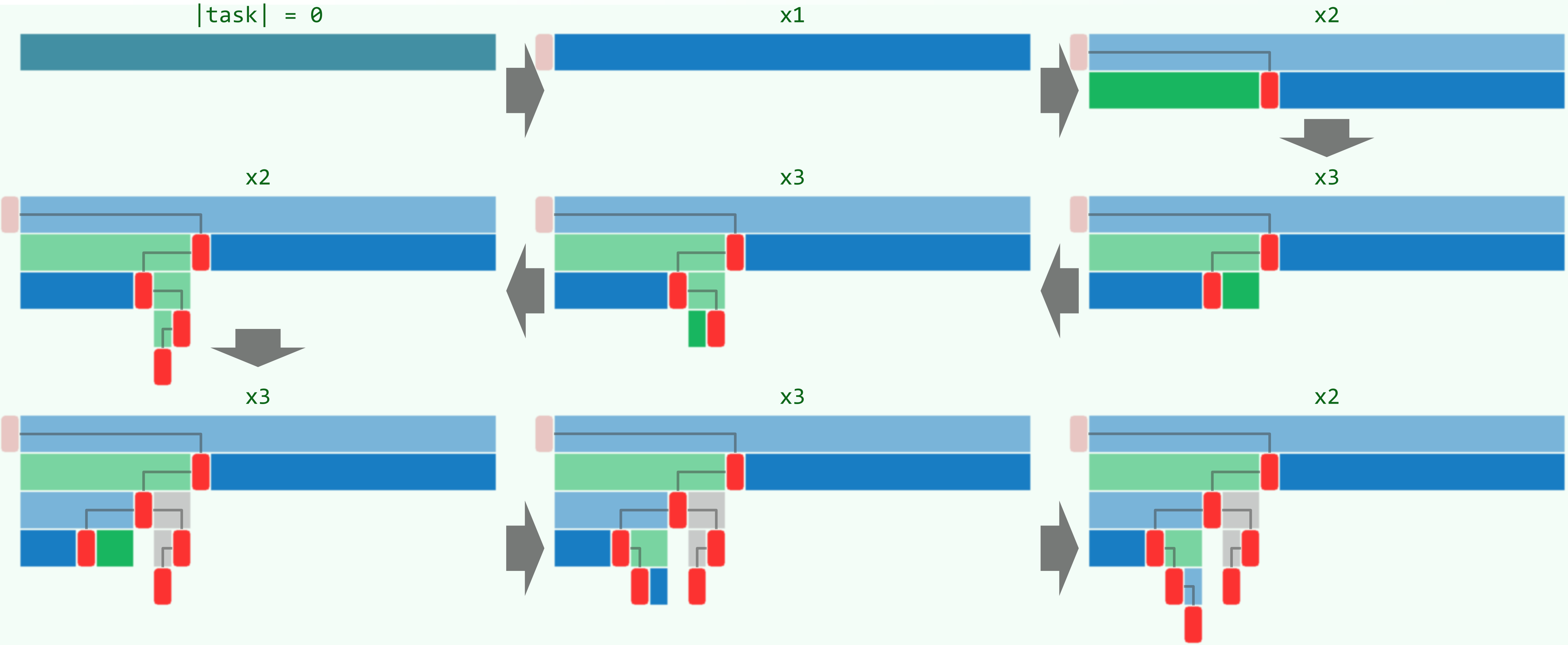


# 迭代化 + 贪心

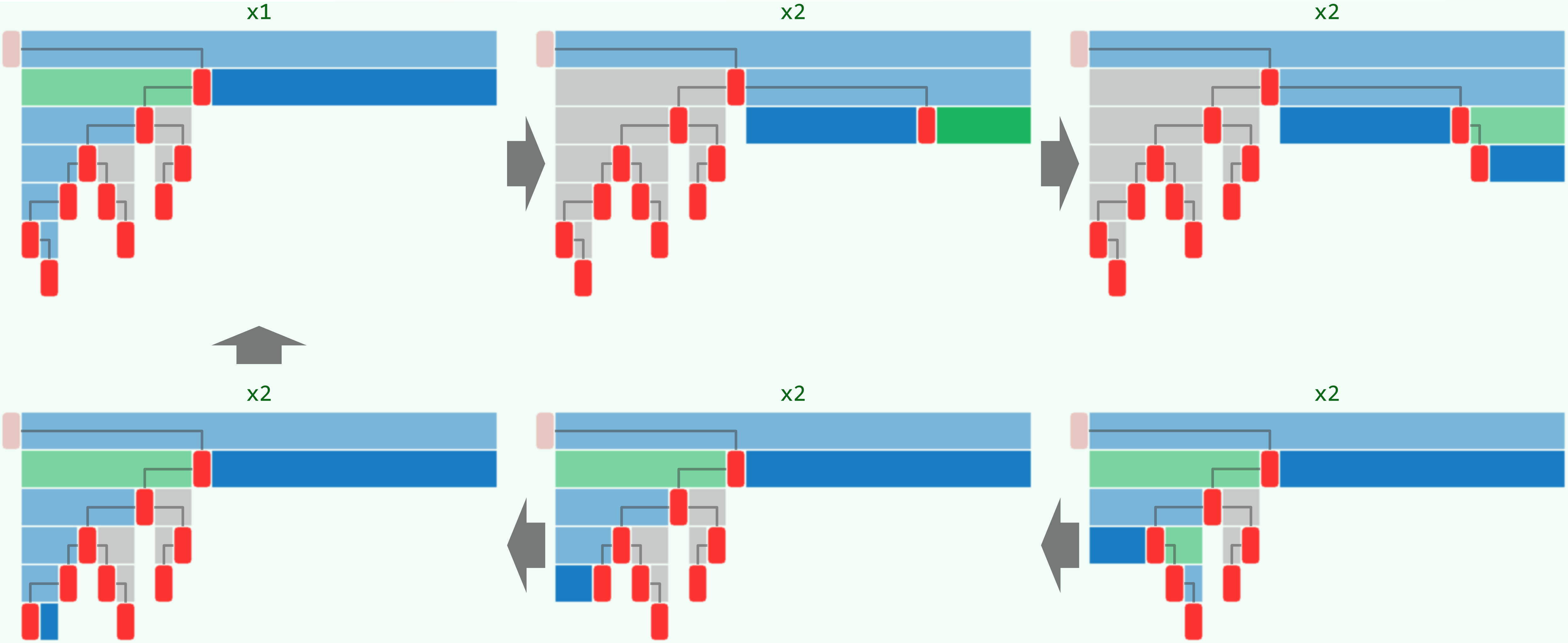
```
#define Put( K, s, t ) { if ( 1 < (t) - (s) ) { K.push(s); K.push(t); } }

#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)
}
```

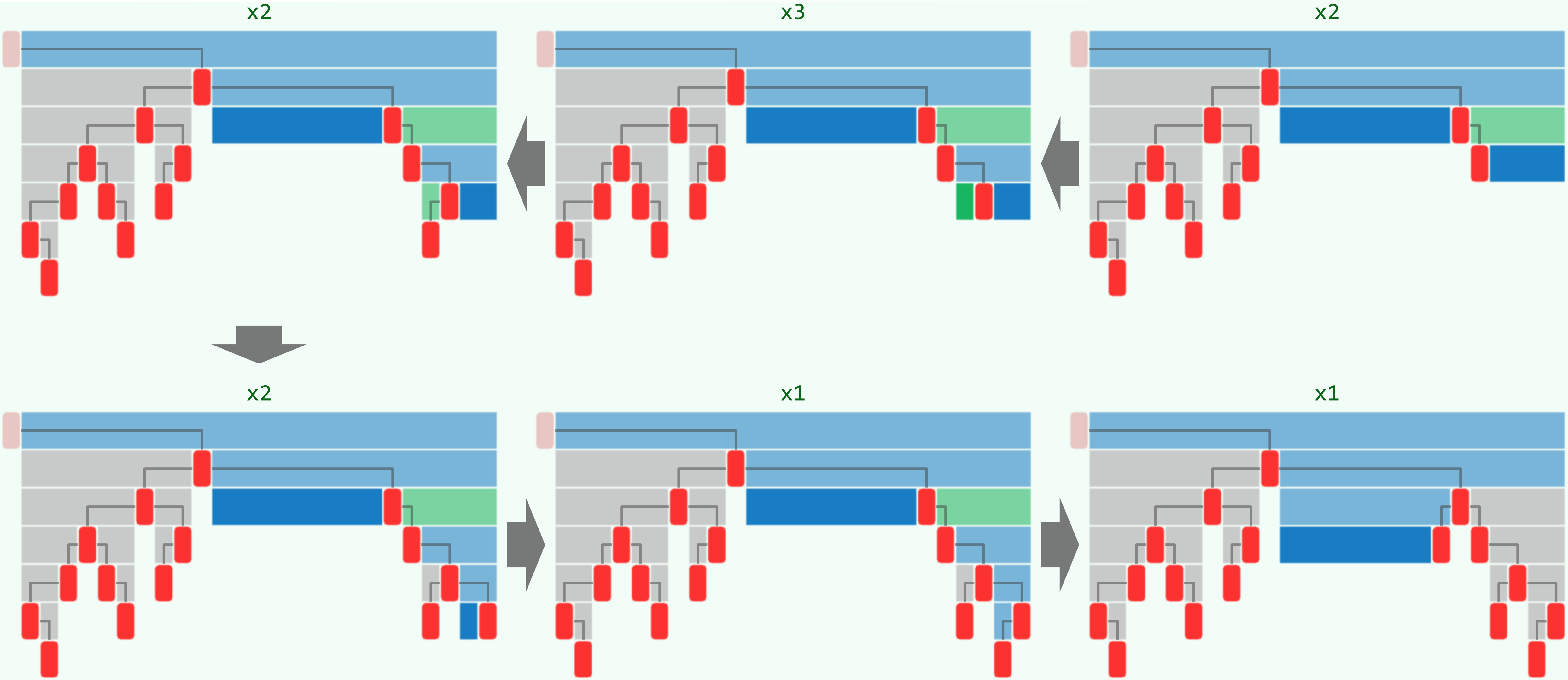
# 实例：0 ~ 7



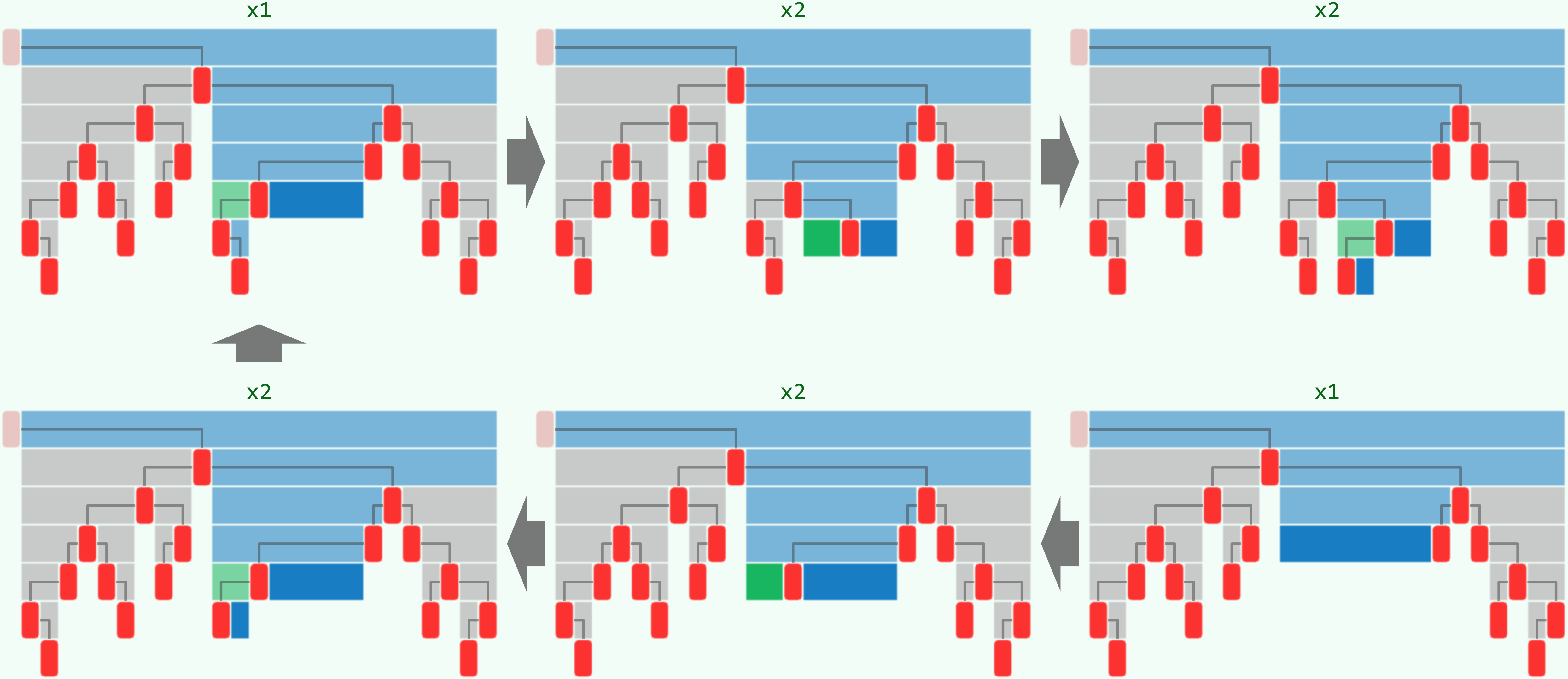
# 实例：7 ~ 12



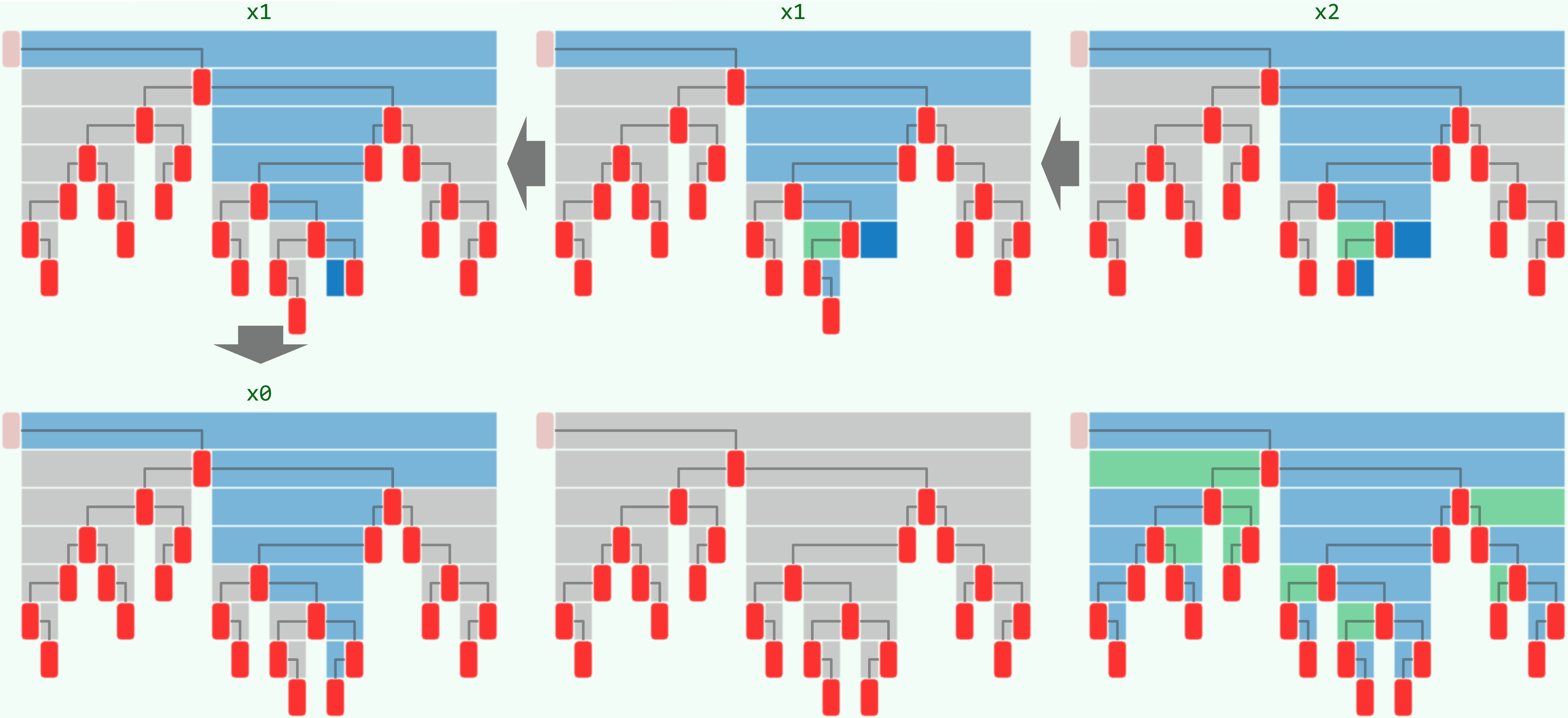
# 实例：12 ~ 17



# 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：$|U| \; = \; u \; \leq \; n - 1 \; < \; n$

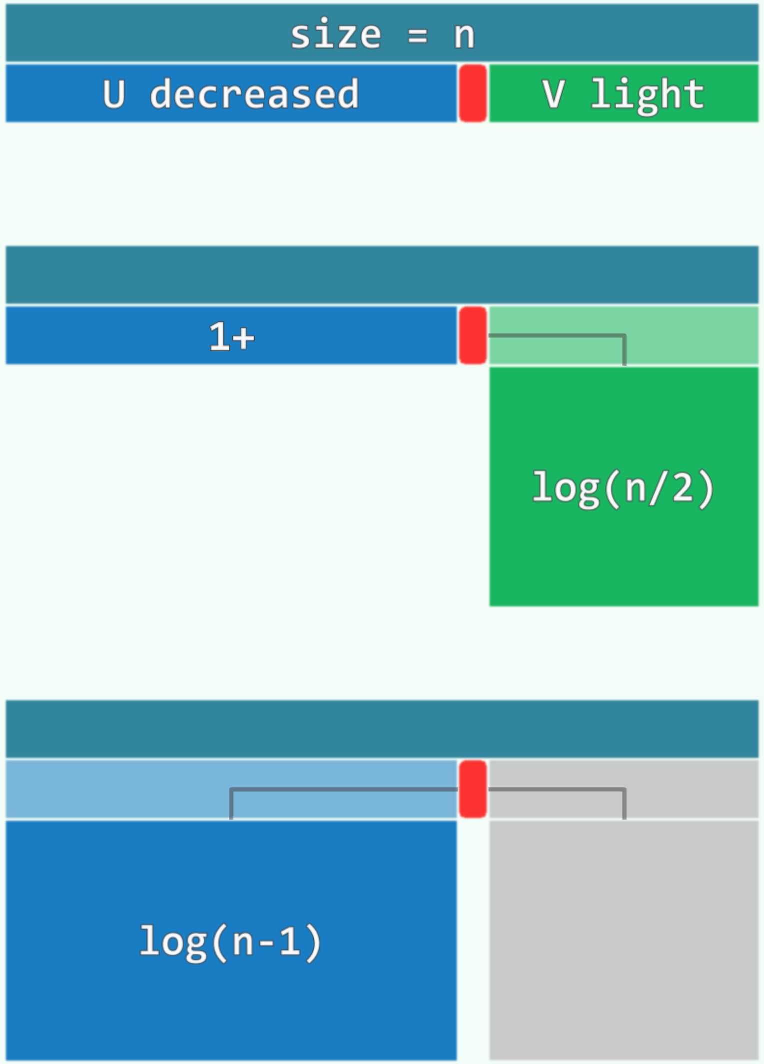
V：接下来，在对 $V$ 的排序（共 $v$ 次划分）过程中

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**

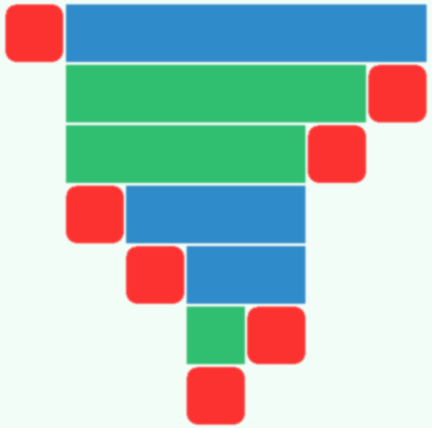
$$T(n) = 2 \cdot T((n-1)/2) + \mathcal{O}(n) = \mathcal{O}(n \cdot \log n)$$

—— 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？