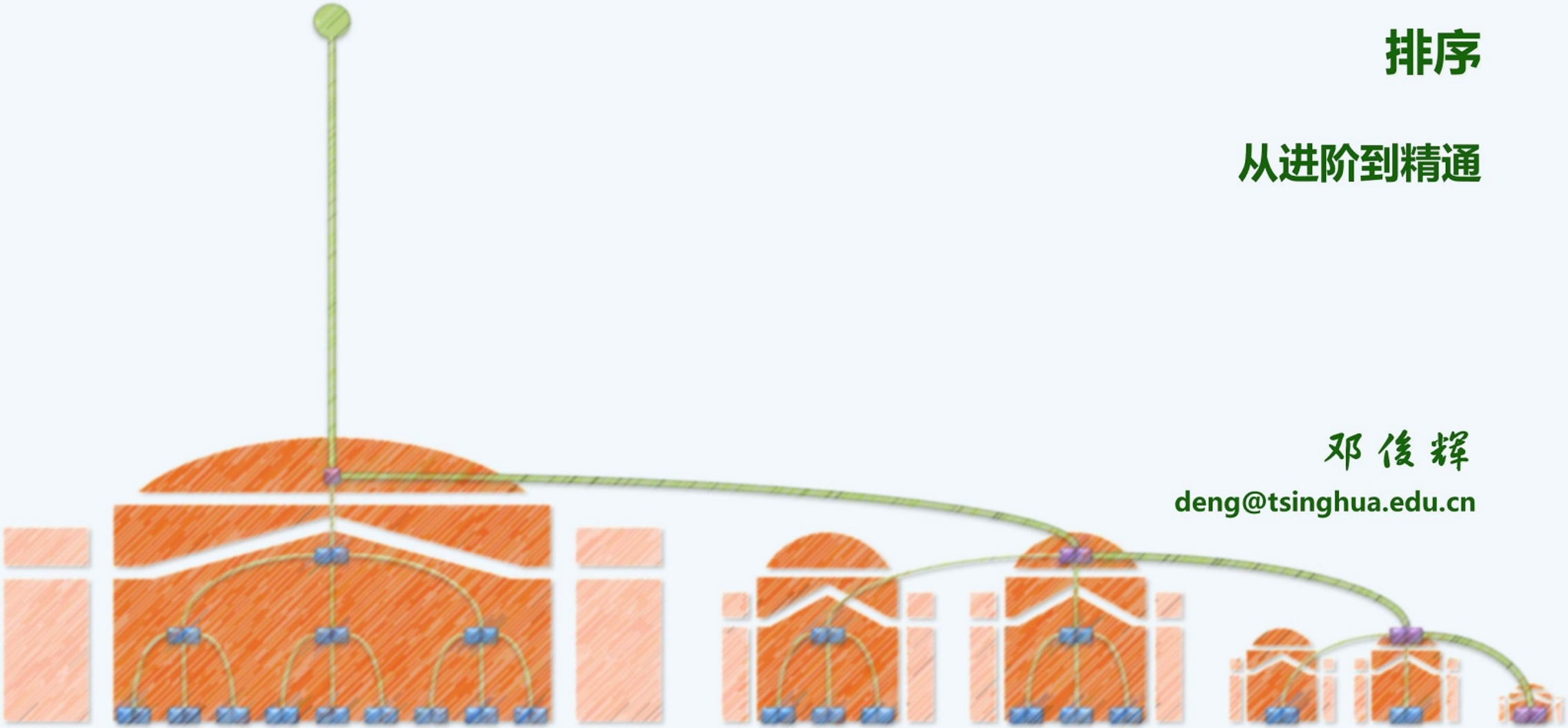
排序

从进阶到精通

邓俊辉

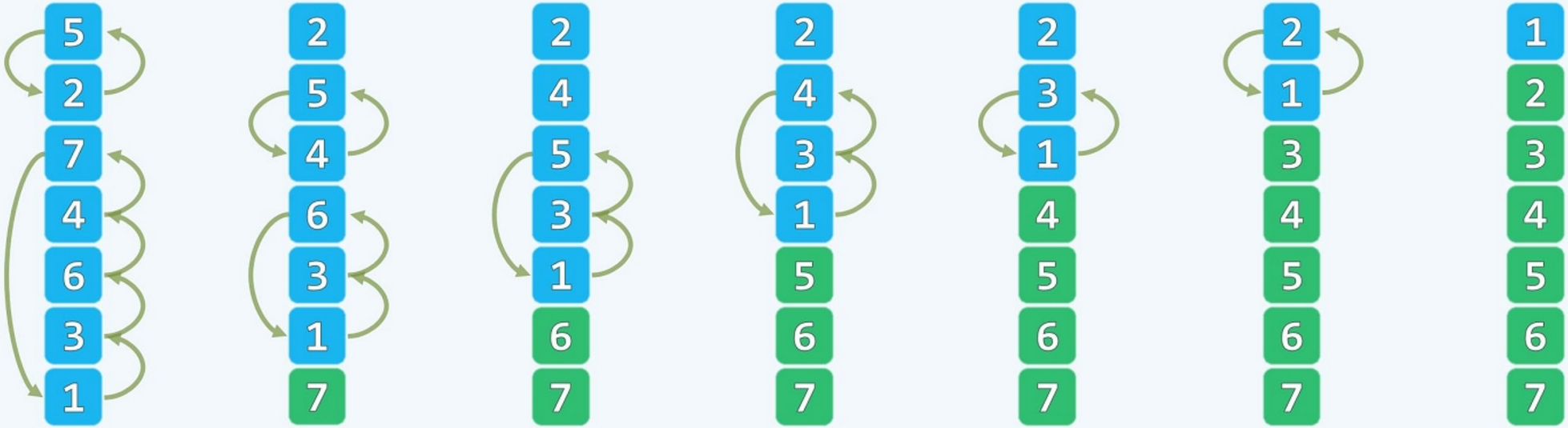
deng@tsinghua.edu.cn



Bubblesort

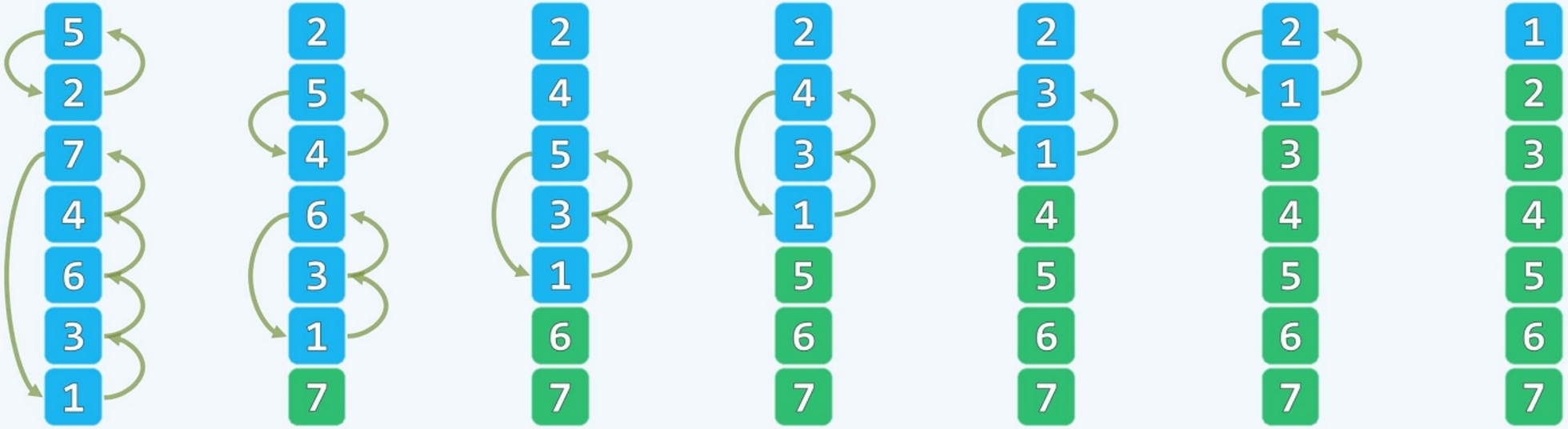
为学日益，为道日损

构思

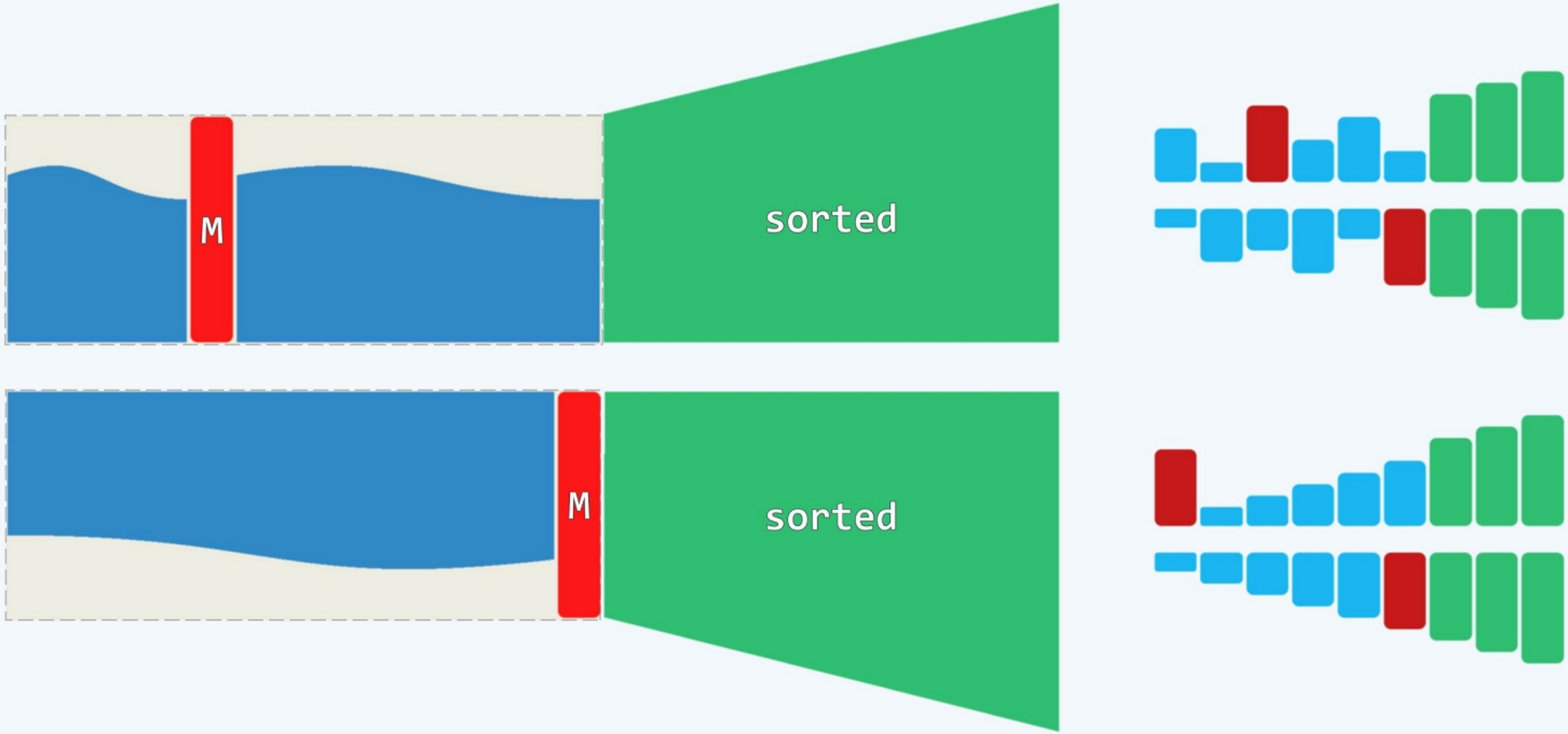


基本版

```
template <typename T> Vector<T>::bubbleSort( Rank lo, Rank hi ) {
    while( lo < --hi ) //逐趟起泡扫描（输入保证：0 <= lo < hi <= size）
        for( Rank i = lo; i < hi; i++ ) //若相邻元素
            if( _elem[i] > _elem[i + 1] ) //逆序
                swap( _elem[i], _elem[i + 1] ); //则交换
}
```

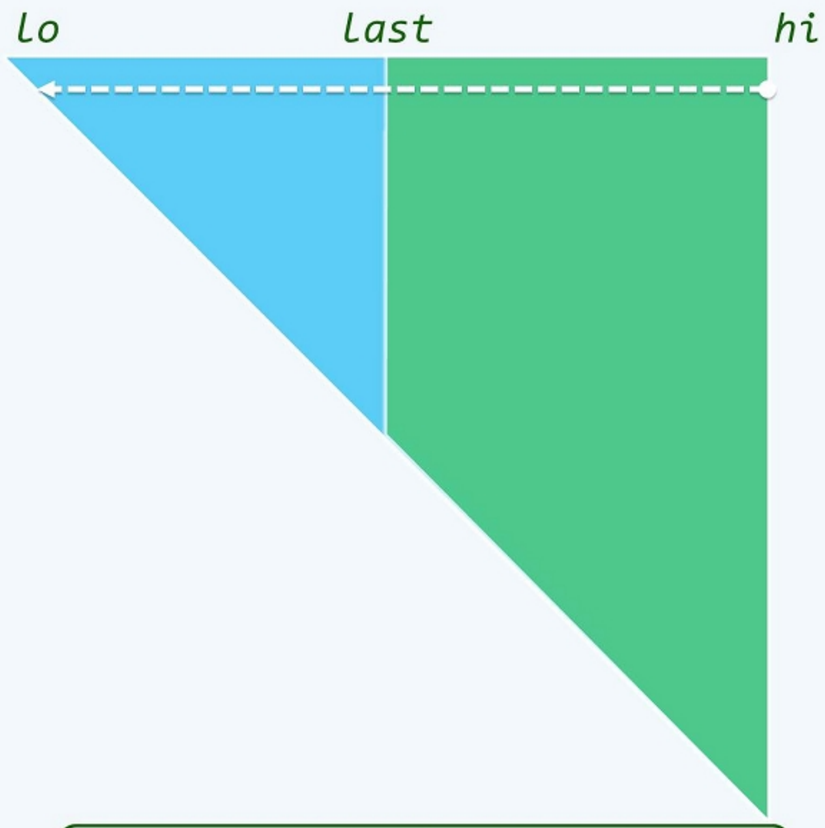


不变性 + 单调性 = 正确性



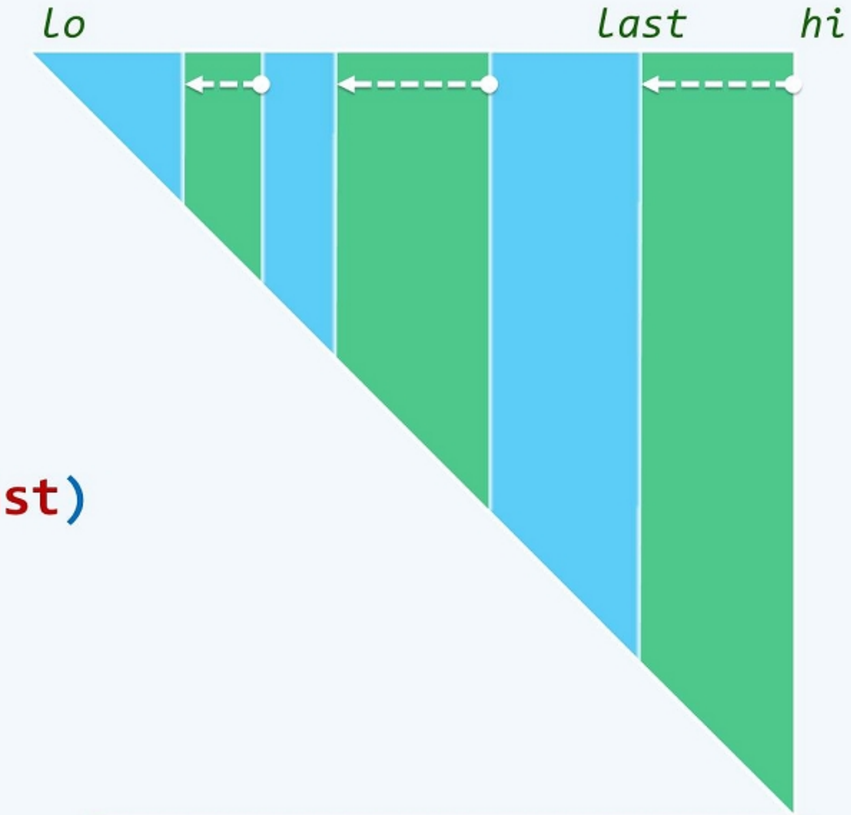
改进：提前终止

```
❖ template <typename T> void Vector<T>::bubbleSort( Rank lo, Rank hi ) {
    for( bool sorted = false; sorted = !sorted; )
       for( Rank i = lo; i < hi - 1; i++ )
          if( _elem[i] > _elem[i + 1] ) {
             swap( _elem[i], _elem[i+1] );
             sorted = false; //仍未完全有序
          } //else ... 提前终止
  }
```



改进：提前终止 + 跳跃

```
❖ template <typename T> void Vector<T>::bubbleSort( Rank lo, Rank hi ) {
    for( Rank last = --hi; lo < hi; hi = last )
       for( Rank i = last = lo; i < hi; i++ )
          if( _elem[i] > _elem[i + 1] ) {
             swap( _elem[i], _elem[i+1] );
             last = i; //逆序对只可能残留于[lo,last)
          }
  }
```



Big-O Notation

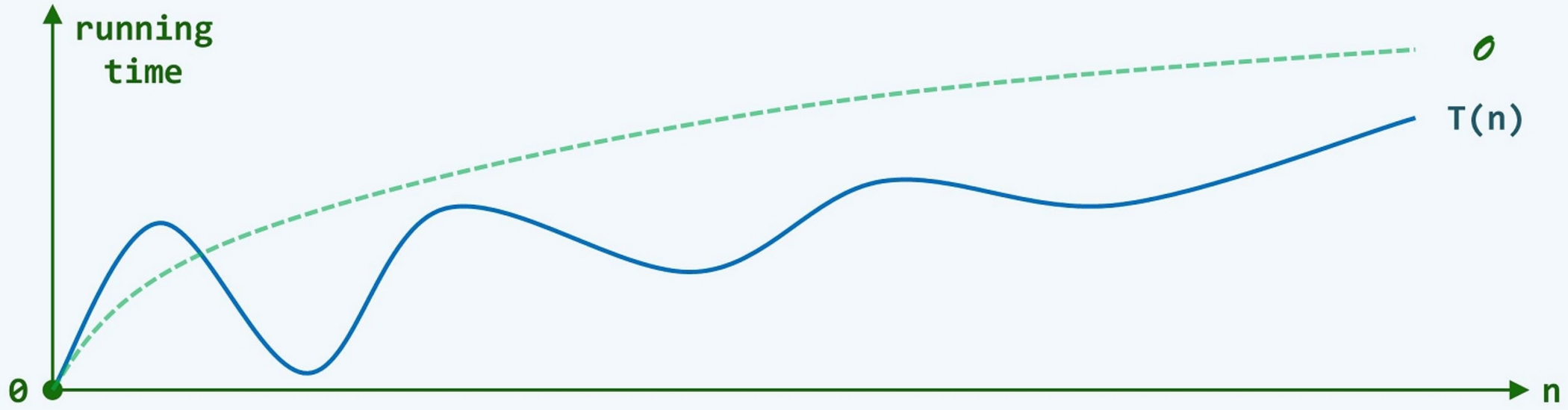
Mathematics is more in need of good notations than of new theorems. - A. Turing

O

$$T(n) = \mathcal{O}(f(n)) \quad \text{iff} \quad \exists\, c > 0 \quad \text{s.t.} \quad T(n) < c \cdot f(n) \quad \forall\, n \gg 2$$

$$Ex: \sqrt{5n \cdot [\underline{3n \cdot (n+2)} + 4] + 6} < \sqrt{\underline{5n \cdot [\underline{6n^2} + 4]} + 6} < \sqrt{\underline{\underline{35n^3} + 6}} < 6 \cdot n^{1.5} = \mathcal{O}(n^{1.5})$$

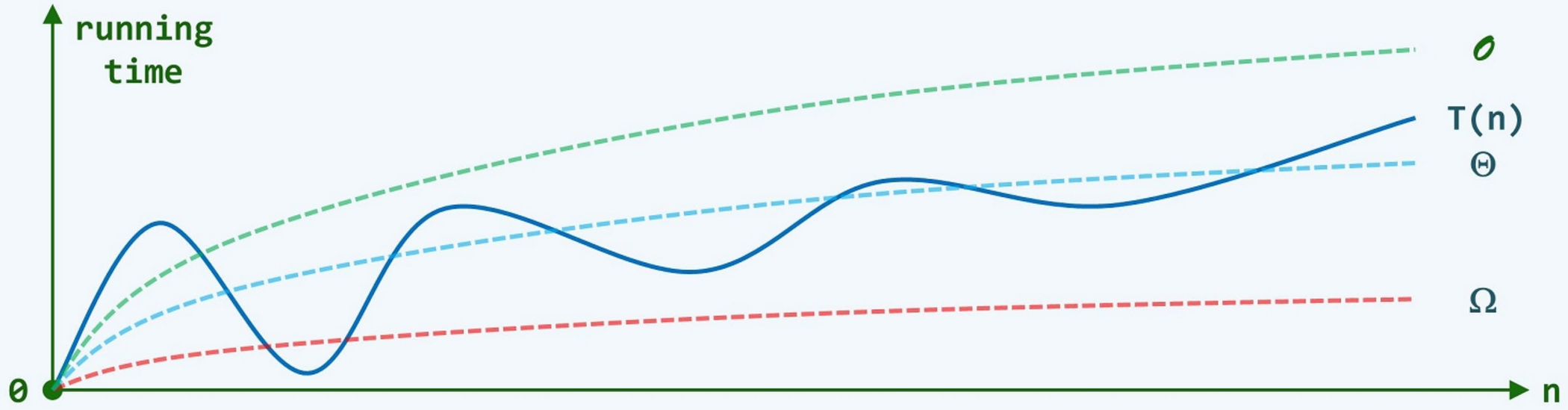
$$\mathcal{O}(f(n)) = \mathcal{O}(c \cdot f(n)) \qquad \mathcal{O}(n^a + n^b) = \mathcal{O}(n^a),\ a \geq b > 0$$



Ω & Θ

$$T(n) = \Omega(f(n)) \quad \text{iff} \quad \exists\, c > 0 \quad \text{s.t.} \quad T(n) > c \cdot f(n) \quad \forall\, n \gg 2$$

$$T(n) = \Theta(f(n)) \quad \text{iff} \quad \exists\, c_1 > c_2 > 0 \quad \text{s.t.} \quad c_1 \cdot f(n) > T(n) > c_2 \cdot f(n) \quad \forall\, n \gg 2$$



复杂度级别

常数	$\mathcal{O}(1)$	再好不过，但难得如此幸运	对数据结构的基本操作
	$\mathcal{O}(\log^* n)$	在这个宇宙中，几乎就是常数	逆Ackermann函数
对数	$\mathcal{O}(\log n)$	与常数无限接近，且不难遇到	有序向量的二分查找 堆、词典的查询、插入与删除
线性	$\mathcal{O}(n)$	努力目标，经常遇到	树、图的遍历
	$\mathcal{O}(n\log^* n)$	几乎几乎几乎接近线性	某些MST算法
	$\mathcal{O}(n\log\log n)$	几乎接近线性	某些三角剖分算法
	$\mathcal{O}(n\log n)$	最常出现，但不见得最优	排序、EU、Huffman编码
平方	$\mathcal{O}(n^2)$	所有输入对象两两组合	Dijkstra算法
立方	$\mathcal{O}(n^3)$	不常见	矩阵乘法
多项式	$\mathcal{O}(n^c)$	P问题 = 存在多项式算法的问题	
指数	$\mathcal{O}(2^n)$	很多问题的平凡算法，再尽可能优化	
...		绝大多数问题，并不存在算法	

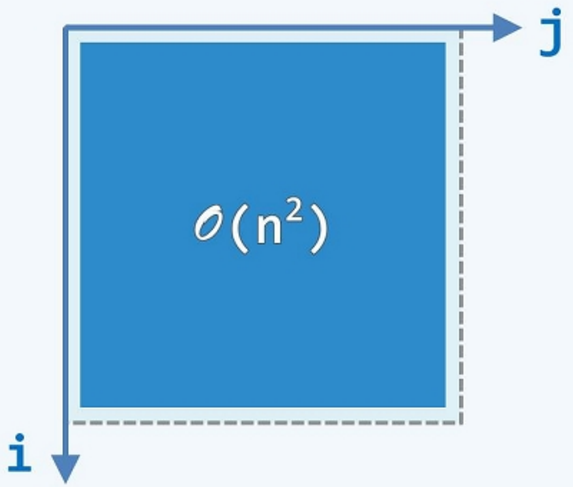
Asymptotic Analysis

谁校对时间，谁就会突然老去。

迭代 + 算术级数

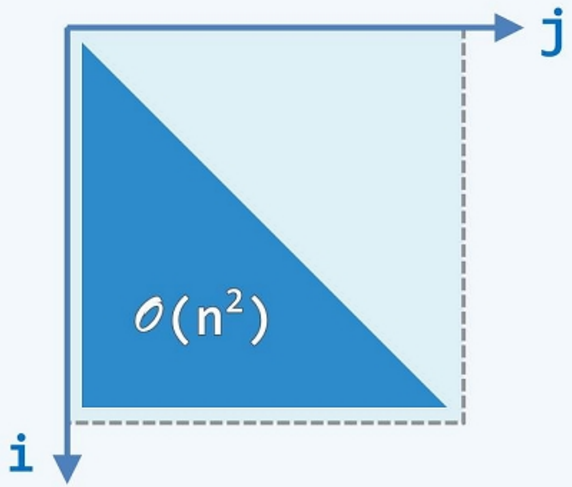
❖
```
for( int i = 0; i < n; i++ )
  for( int j = 0; j < n; j++ )
    O1op(const i, const j);
```

$$\sum_{i=0}^{n-1} n = n \times n = \mathcal{O}(n^2)$$



❖
```
for( int i = 0; i < n; i++ )
  for( int j = 0; j < i; j++ )
    O1op(const i, const j);
```

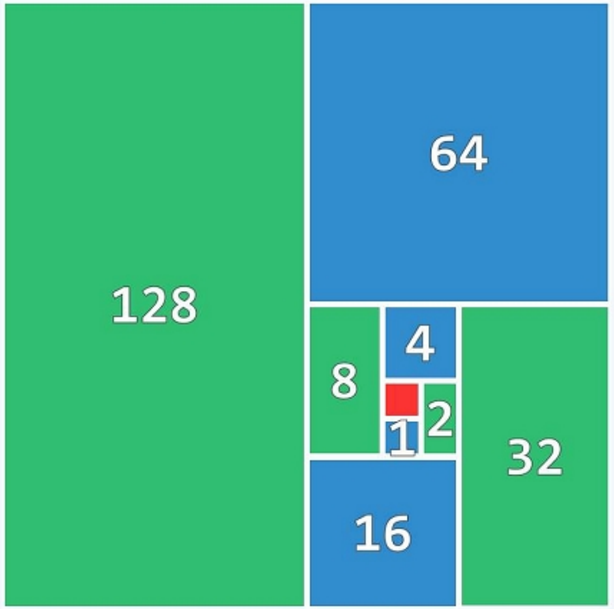
$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$$



迭代 vs. 级数

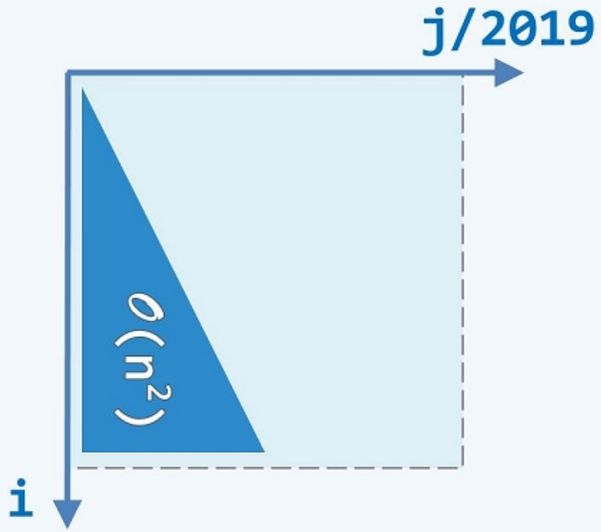
```
❖ for( int i = 1; i < n; i <<= 1 )
  for( int j = 0; j < i; j++ )
    O1op(const i, const j);
```

```
❖ for( int i = 0; i < n; i++ )
  for( int j = 0; j < i; j += 2019 )
    O1op(const i, const j);
```



$$1 + 2 + 4 + \cdots + 2^{\lfloor \log_2 (n-1) \rfloor}$$

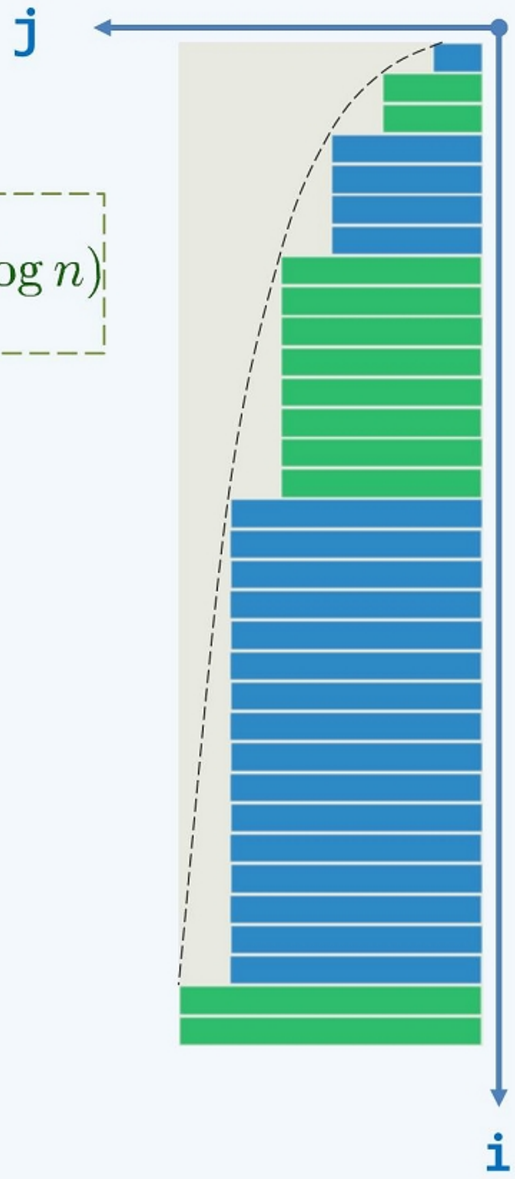
$$= 2^{\lceil \log_2 n \rceil} - 1 = \mathcal{O}(n)$$



迭代 + 复杂级数

❖
```
for( int i = 0; i <= n; i++ )
  for( int j = 1; j < i; j += j )
    O1op(const i, const j);
```

$$T(n) = \sum_{i=0}^{n} \lceil \log_2 i \rceil = \mathcal{O}(n \log n)$$



$$T(n) \approx \int_{x=0}^{n} \ln(x)dx = n \cdot \ln(n) - \int_{x=0}^{n} x \cdot d(\ln(x)) = n \cdot \ln(n) - n$$

$$T(n) = \sum_{k=1}^{\log n} k \cdot 2^{k-1} = \sum_{k=1}^{\log n} \sum_{i=1}^{k} 2^{k-1} = \sum_{i=1}^{\log n} \sum_{k=i}^{\log n} 2^{k-1}$$

$$T(n) \le \sum_{i=1}^{\log n} \sum_{k=1}^{\log n} 2^{k-1} \le \sum_{i=1}^{\log n} 2^{\log n} = \sum_{i=1}^{\log n} n = n \cdot \log n$$

$$T(n) \ge \sum_{i=1}^{\log n} \sum_{k=\log n}^{\log n} 2^{k-1} \ge \sum_{i=1}^{\log n} 2^{\log n - 1} = \sum_{i=1}^{\log n} n/2 = n/2 \cdot \log n$$

Back Of Envelope Calculation

He calculated just as men breathe, as eagles sustain themselves in the air.

Back-Of-The-Envelope Calculation

- 地球（赤道）周长 ≈ 787 x 360/7.2

  = 787 x 50 = 39,350 km

- 1天 = 24hr x 60min x 60sec

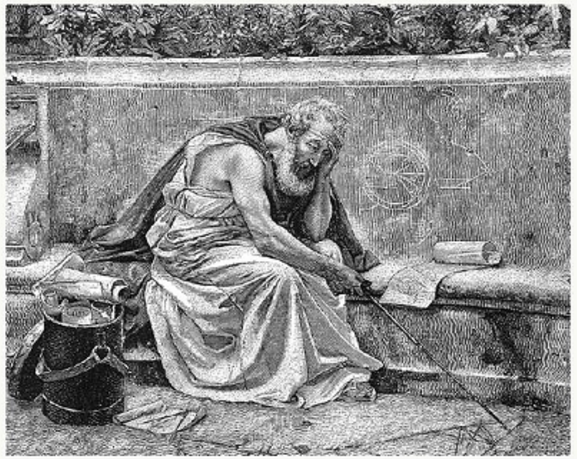
  ≈ 25 x 4000 = 10^5 sec

- 1生 ≈ 1世纪 = 100yr x 365 = 3 x 10^4 day = 3 x 10^9 sec

- “为祖国健康工作五十年” ≈ 1.6 x 10^9 sec

- “三生三世” ≈ 300 yr = 10^10 = (1 googel)^(1/10) sec

- 宇宙大爆炸至今 = 4 x 10^17 > 10^8 x 一生



Eratosthenes
(276 ~ 194 B.C.)

Enrico Fermi
(1901 - 1954)

实例：全国人口普查数据的排序

$n = 1.4^{+} \times 10^{9}$

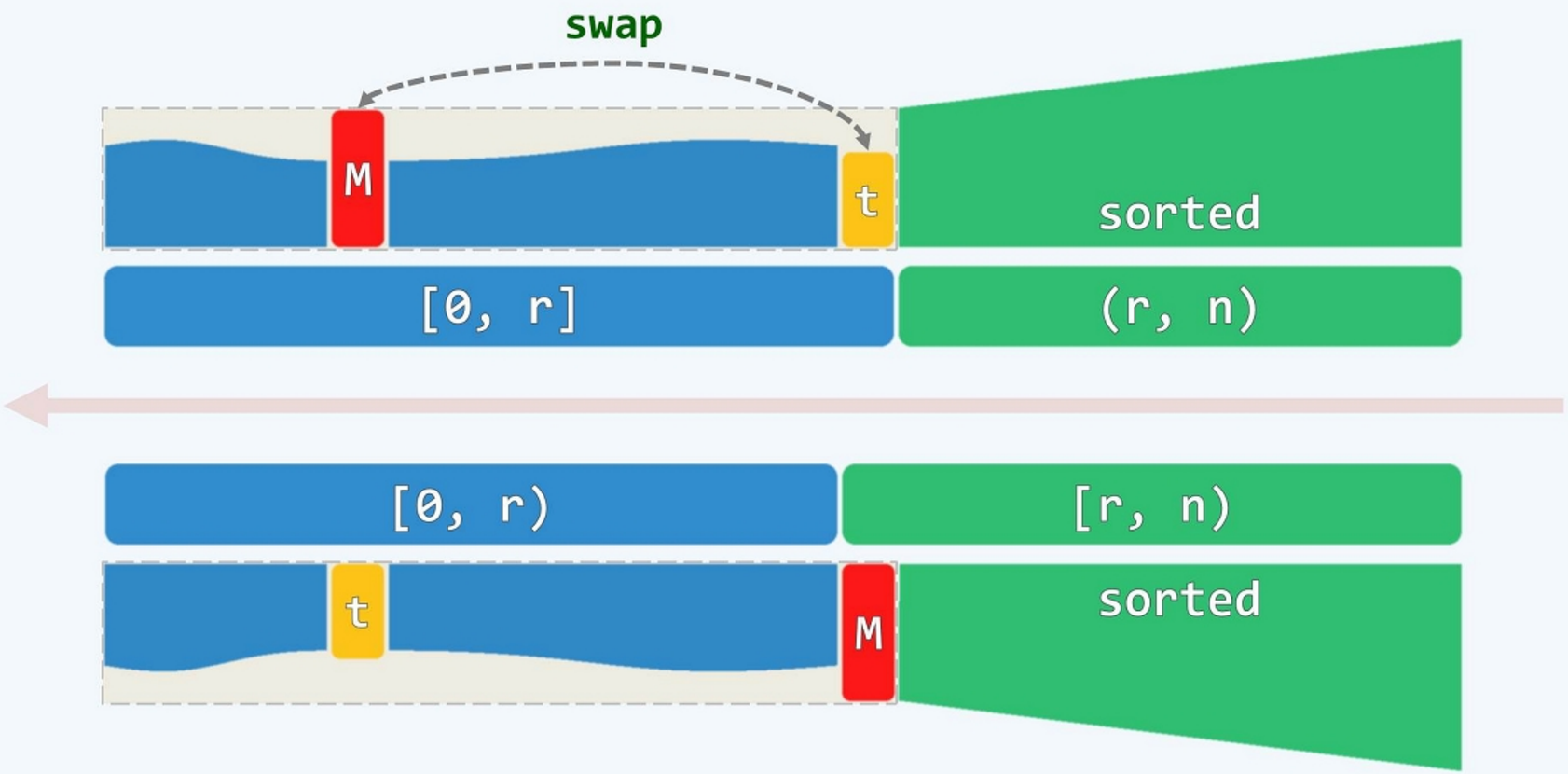
$\mathcal{O}(10^{9})$

硬件 \ 算法	Bubblesort (10^9)^2 10^18	Mergesort 10^9 x log(10^9) 30 x 10^9
普通PC 1GHz 10^9 flops	10^9 sec 30 yr	30 sec
天河1A 千万亿次 = 1P 10^15 flops	10^3 sec 20 min	0.03 ms

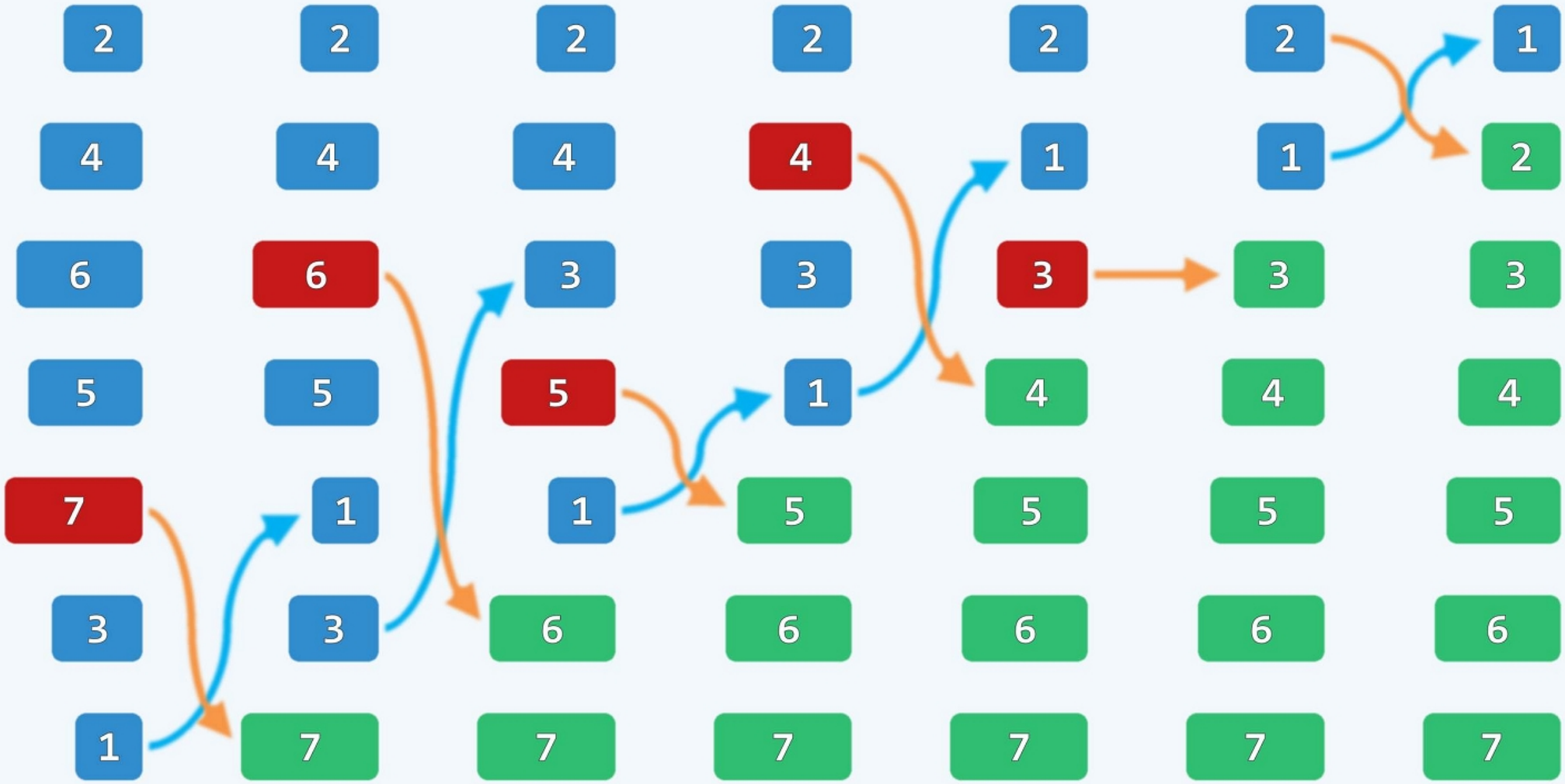
Selectionsort

天下只有两种人。譬如一串葡萄到手，一种人挑最好的先吃，另一种人把最好的留在最后吃。

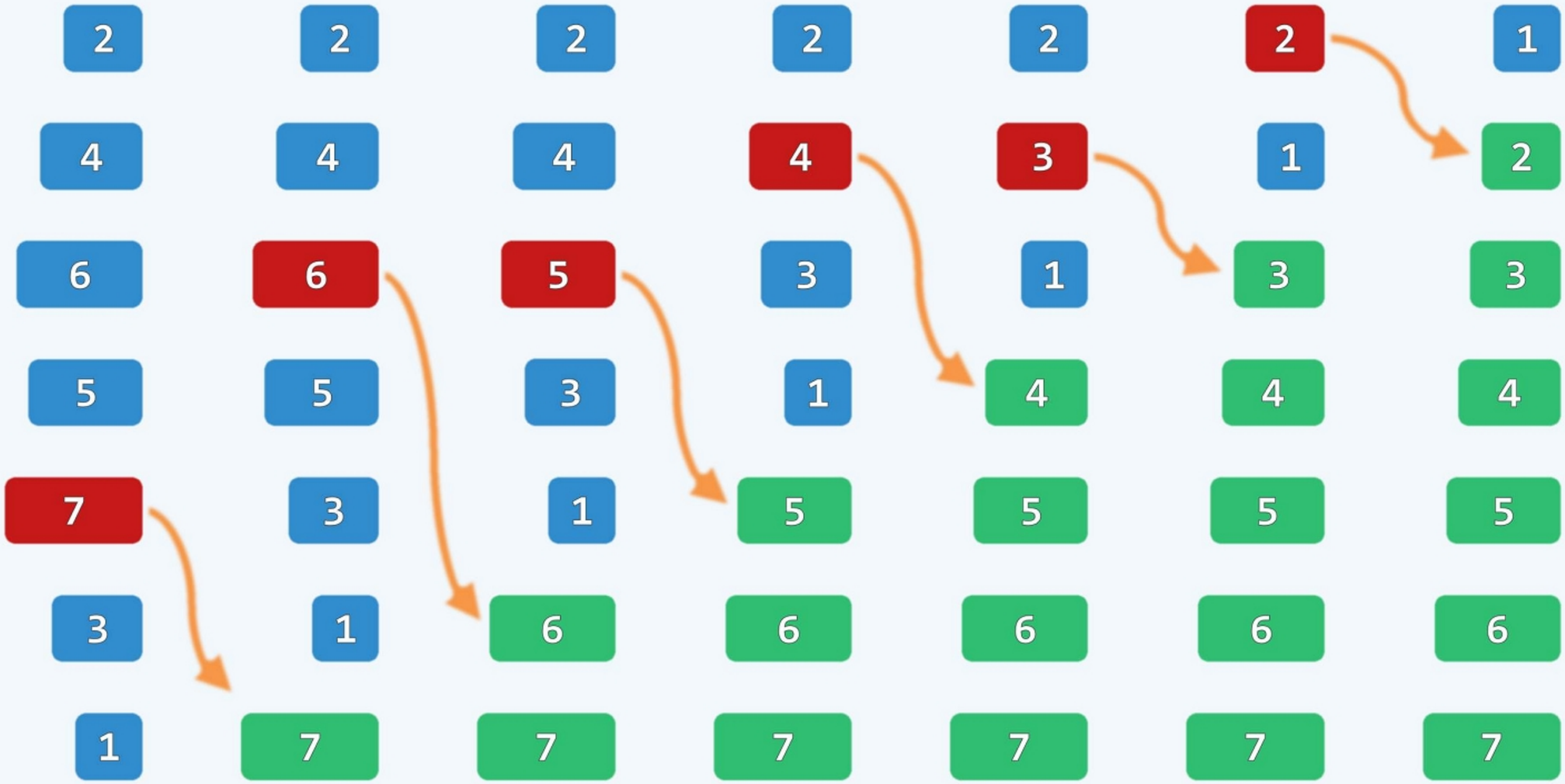
起泡排序，温故知新



交换法

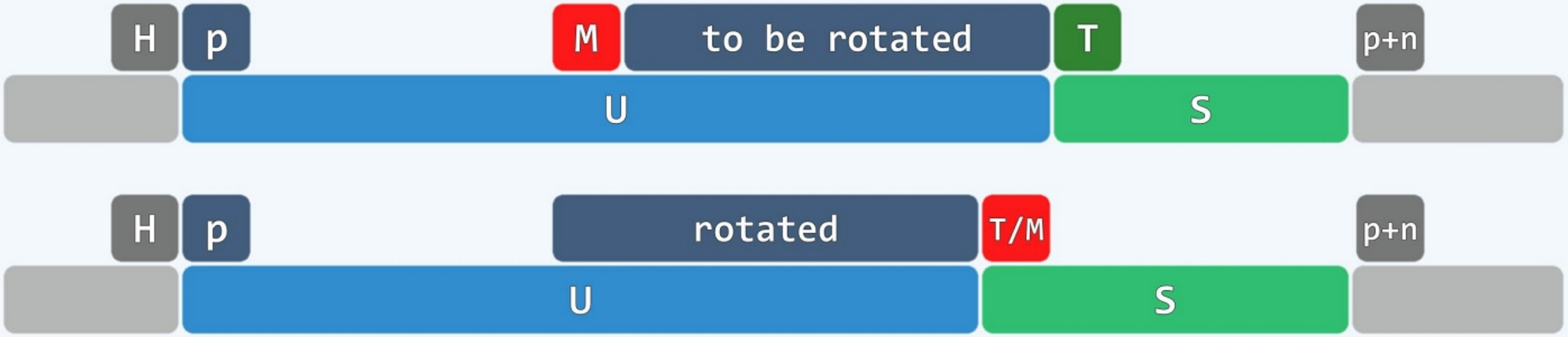


平移法



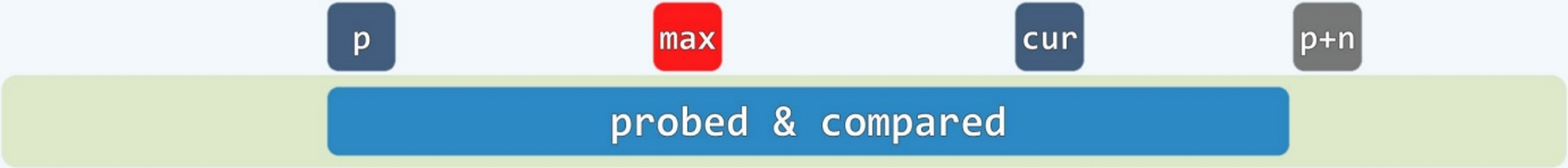
selectionSort()

```
//对列表中起始于位置p的连续n个元素做选择排序，valid(p) && rank(p) + n <= size
  template <typename T> void List<T>::selectionSort( Posi(T) p, int n ) {
     Posi(T) head = p->pred; Posi(T) tail = p; //待排序区间(head, tail)
     for ( int i = 0; i < n; i++ ) tail = tail->succ; //head/tail可能是头/尾哨兵
     while ( 1 < n ) { //反复从（非平凡）待排序区间内找出最大者，并移至有序区间前端
        insertB( tail, remove( selectMax( head->succ, n ) ) ); //改进...
        tail = tail->pred; n--; //待排序区间、有序区间的范围，均同步更新
     }
  }
```



selectMax()

```
❖ template <typename T> //从起始于位置p的n个元素中选出最大者，1 < n
  Posi(T) List<T>::selectMax( Posi(T) p, int n ) { //Θ(n)
     Posi(T) max = p; //最大者暂定为p
     for ( Posi(T) cur = p; 1 < n; n-- ) //后续节点逐一与max比较
        if ( ! lt( (cur = cur->succ)->data, max->data ) ) //data ≥ max
           max = cur; //则更新最大元素位置记录
     return max; //返回最大节点位置
  }
```

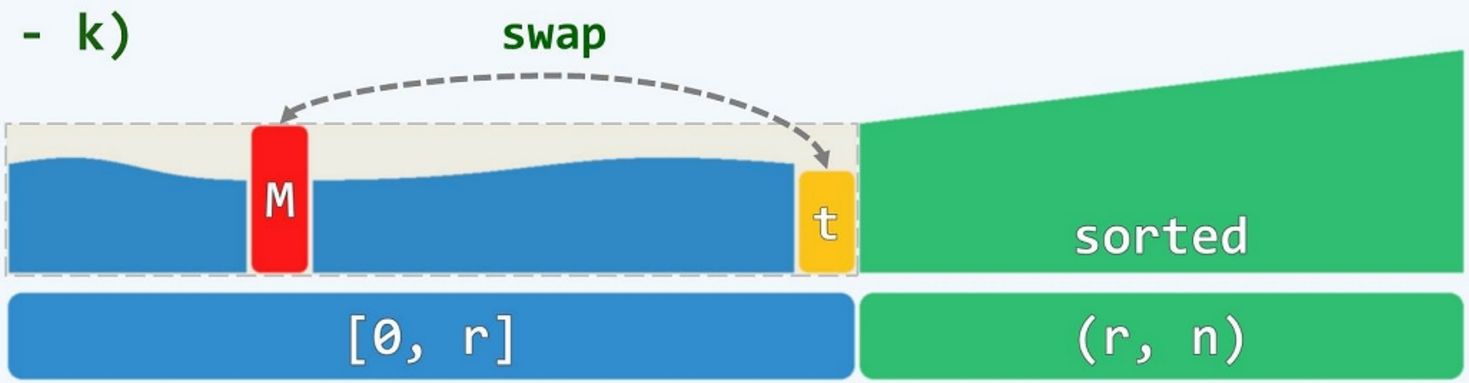


性能分析

❖ 共迭代n次，在第k次迭代中

- selectMax()耗时$\Theta(n - k)$
- swap()耗时$\mathcal{O}(1)$

故总体复杂度为$\Theta(n^2)$



❖ 尽管如此，元素的**移动**操作远远少于起泡排序 //实际更为费时

也就是说，$\Theta(n^2)$主要来自于元素的**比较**操作 //成本相对更低

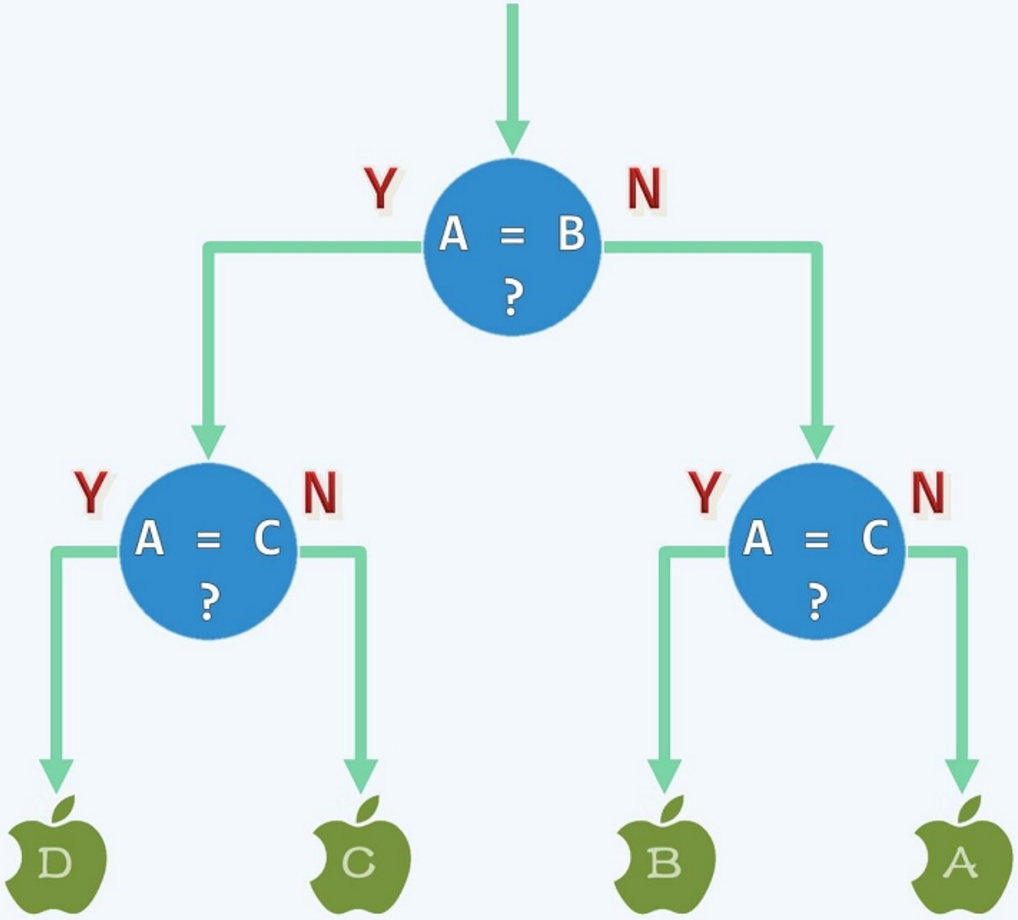
❖ 可否...**更快**地完成排序？首先...**最快**能有多快？

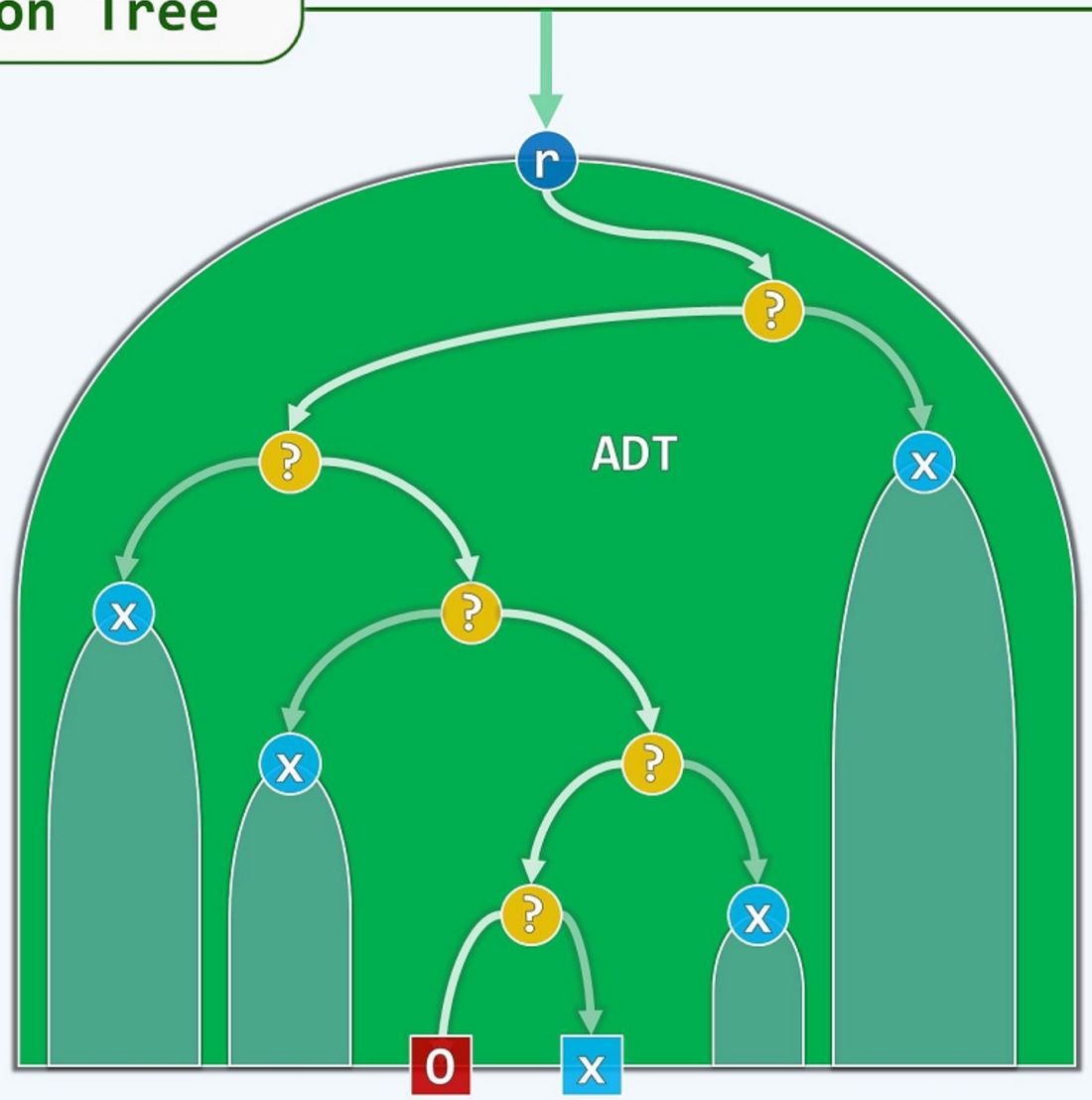
Lower Bound

言有易，言无难

不怕不识货，就怕货比货

谁是坏苹果 -> Algebraic Decision Tree

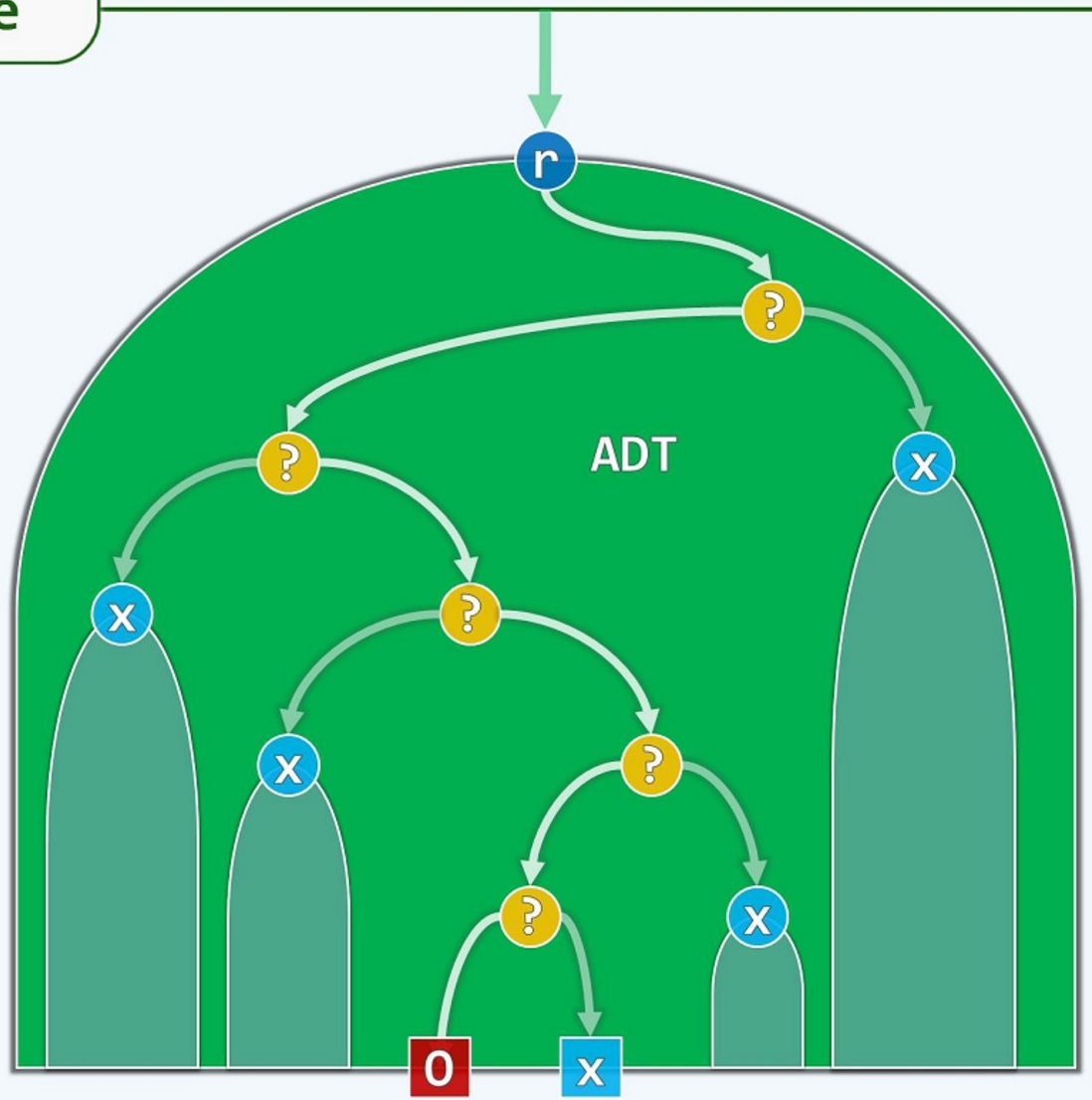




Lower Bound <- Comparison Tree

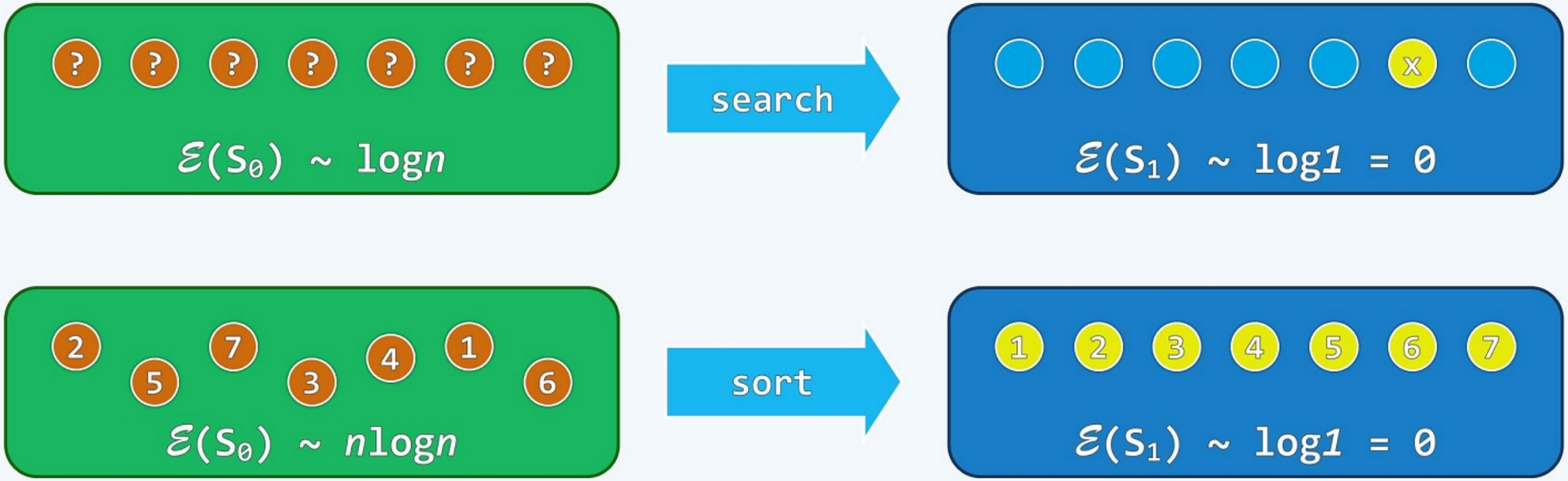
$$\#leaf \;\geq\; n!$$

$$\begin{aligned} height \;&\geq\; \log n! \\ &=\; \log e \cdot (n \ln n - n + \mathcal{O}(\ln n)) \\ &=\; \Omega(n \cdot \log n) \end{aligned}$$



Lower Bound -> Entropy

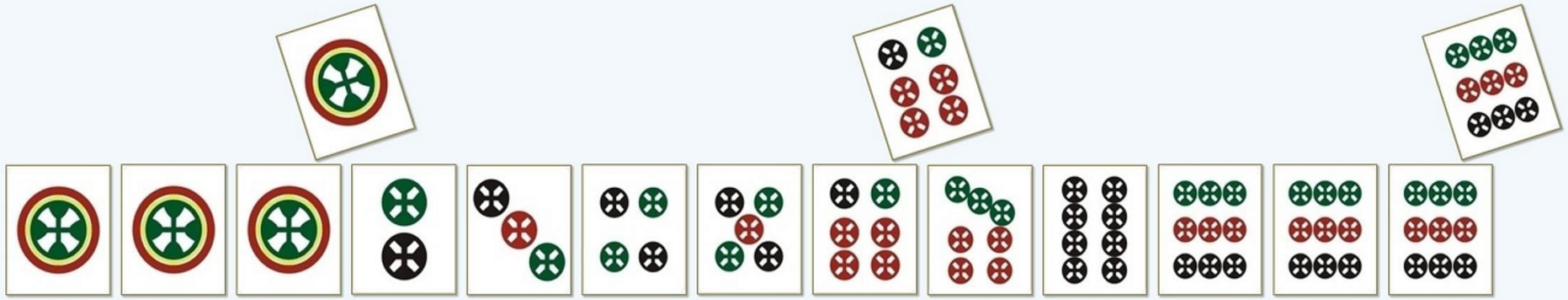
$$\mathcal{E}(S) = \log N = \log n! = \Omega(n \log n)$$



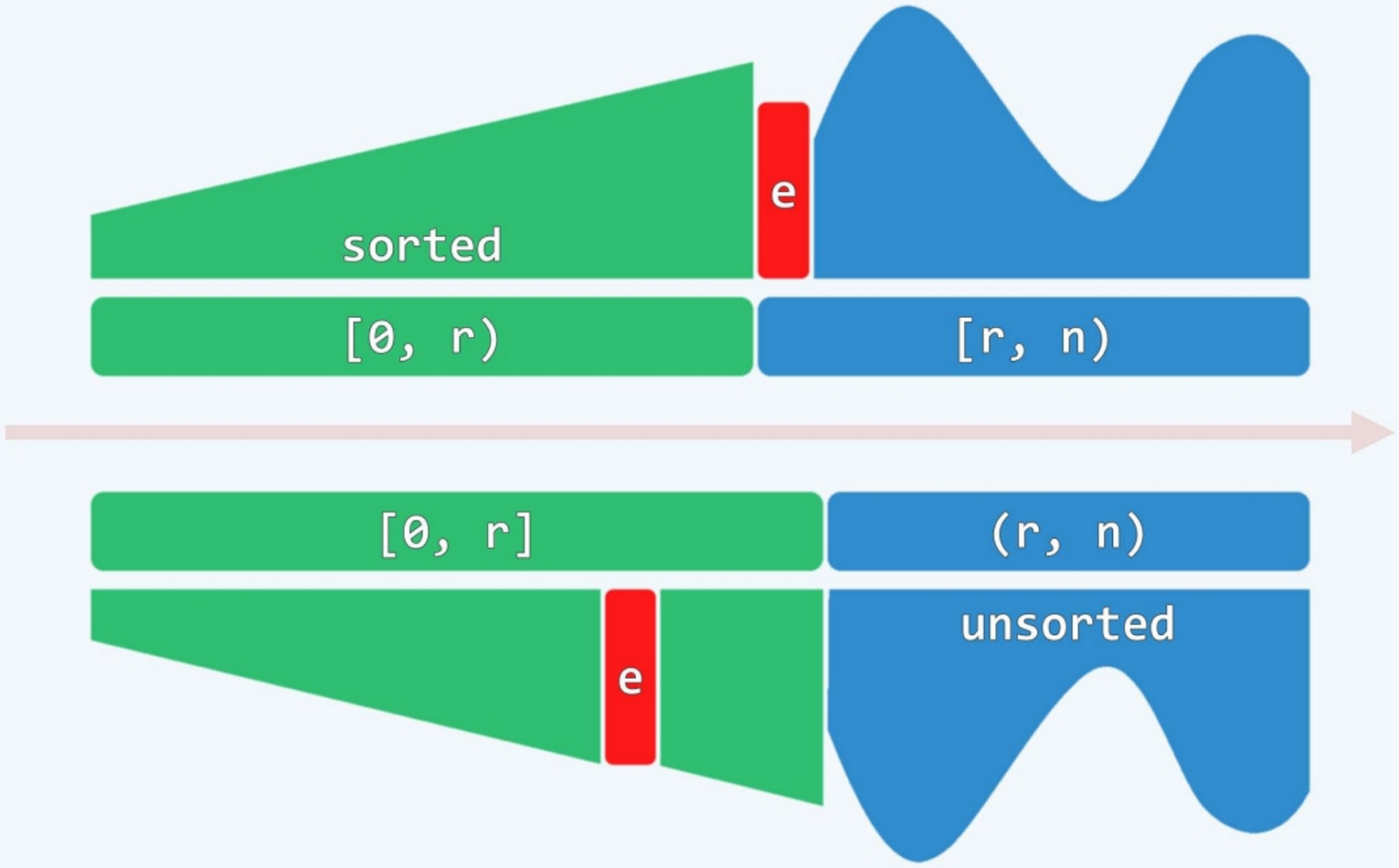
Insertionsort

一语未了，只见宝玉笑嘻嘻的掮了一枝红梅进来，众丫鬟忙已接过，插入瓶内。

学而优则玩



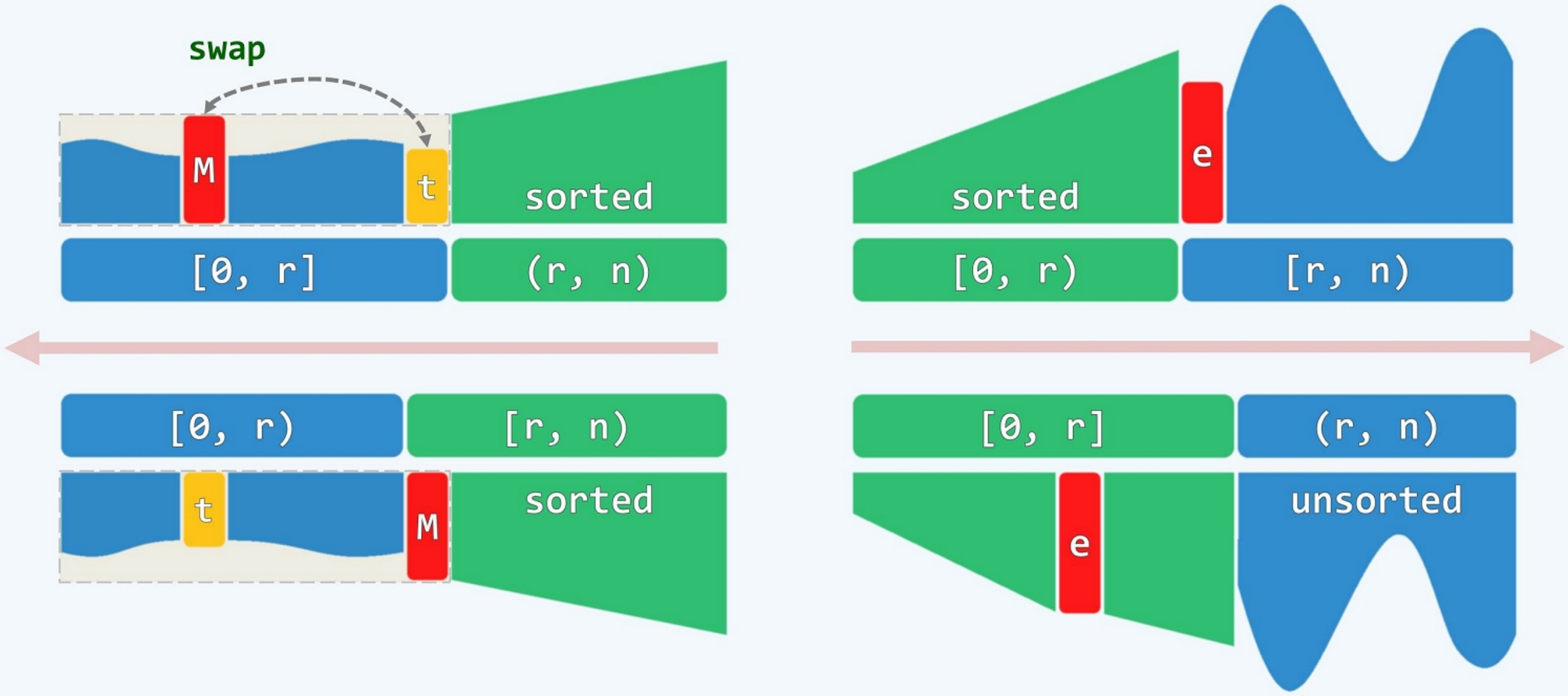
减而治之



实现

```
//对列表中起始于位置p的连续n个元素做插入排序，valid(p) && rank(p) + n <= size
  template <typename T> void List<T>::insertionSort( Posi(T) p, int n ) {
     for ( int r = 0; r < n; r++ ) { //逐一引入各节点，由Sr得到Sr+1
        insertA( search( p->data, r, p ), p->data ); //查找 + 插入
        p = p->succ; remove( p->pred ); //转向下一节点
     } //n次迭代，每次O(r + 1)
  } //仅使用O(1)辅助空间，属于就地算法
```

不怕不识货


swap
M
t
sorted
[0, r]
(r, n)
[0, r)
[r, n)
t
M
sorted
sorted
e
[0, r)
[r, n)
[0, r]
(r, n)
e
unsorted

Inversion

有象斯有對，對必反其為；有反斯有讎，讎必和而解

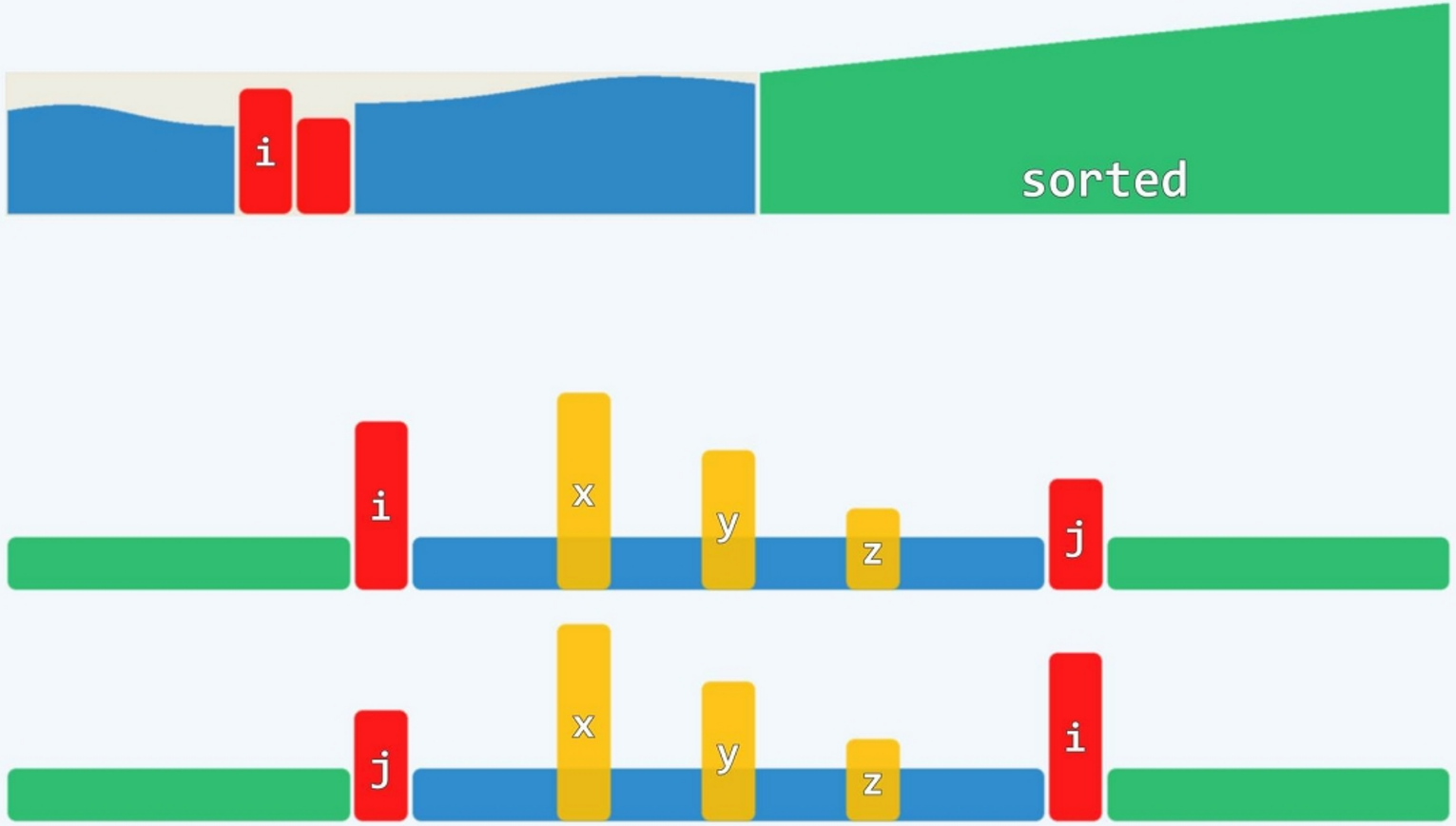
Inversion

$\langle i, j \rangle$ is called an inversion if $0 \le i < j < n$ and $A[i] > A[j]$

$$\mathcal{I}(j) = \left\| \{ \, 0 \le i < j \;\mid\; A[i] > A[j] \text{ and hence } \langle i, j \rangle \text{ is an inversion} \, \} \right\|$$

$$\mathcal{I} = \sum_j \mathcal{I}(j) \le \binom{n}{2} = \mathcal{O}(n^2)$$

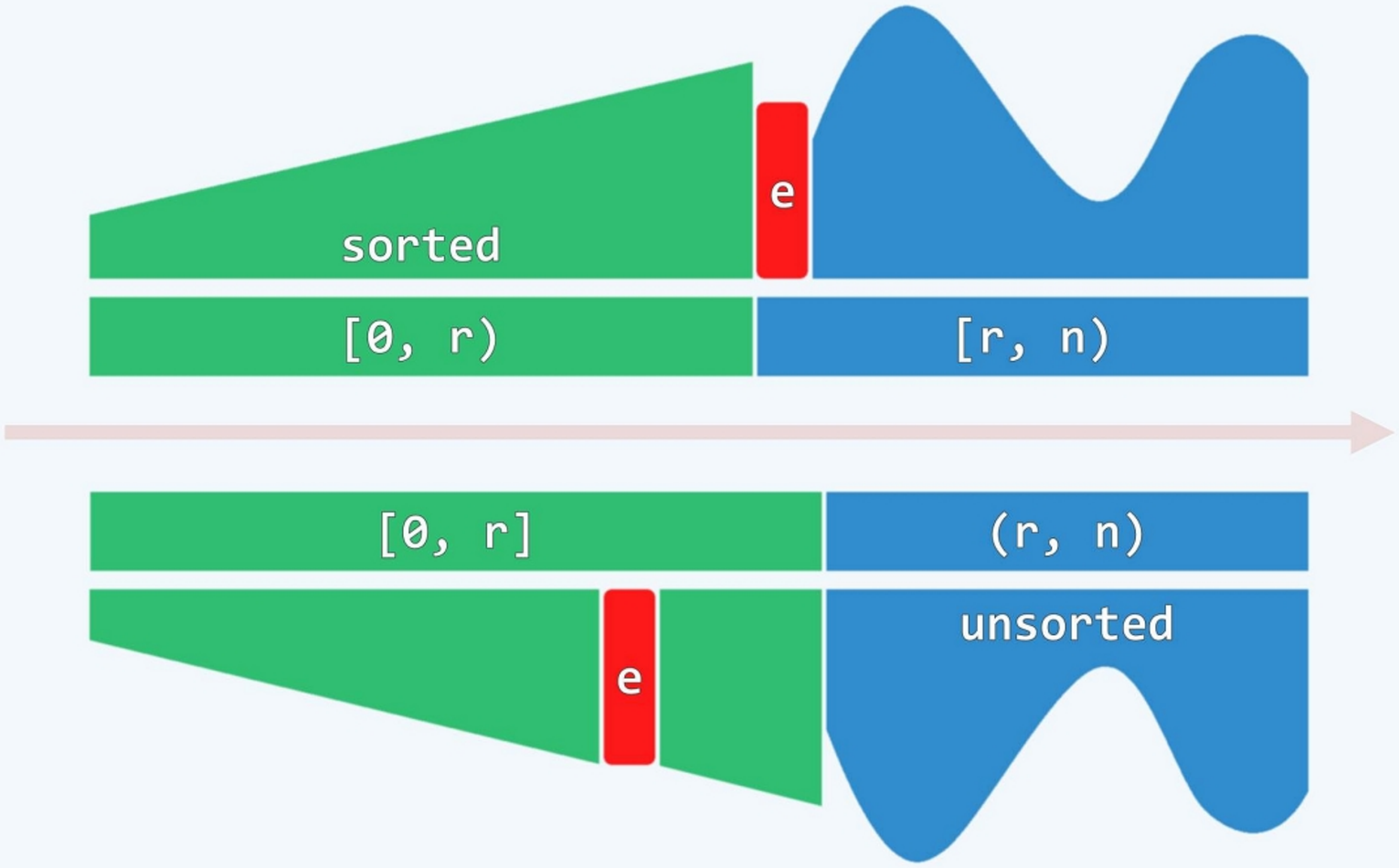
Inversions In Bubblesort



Inversions In Insertionsort

$$|search(r)| = \mathcal{I}(r)$$

$$\sum_r |search(r)| = \mathcal{I}$$



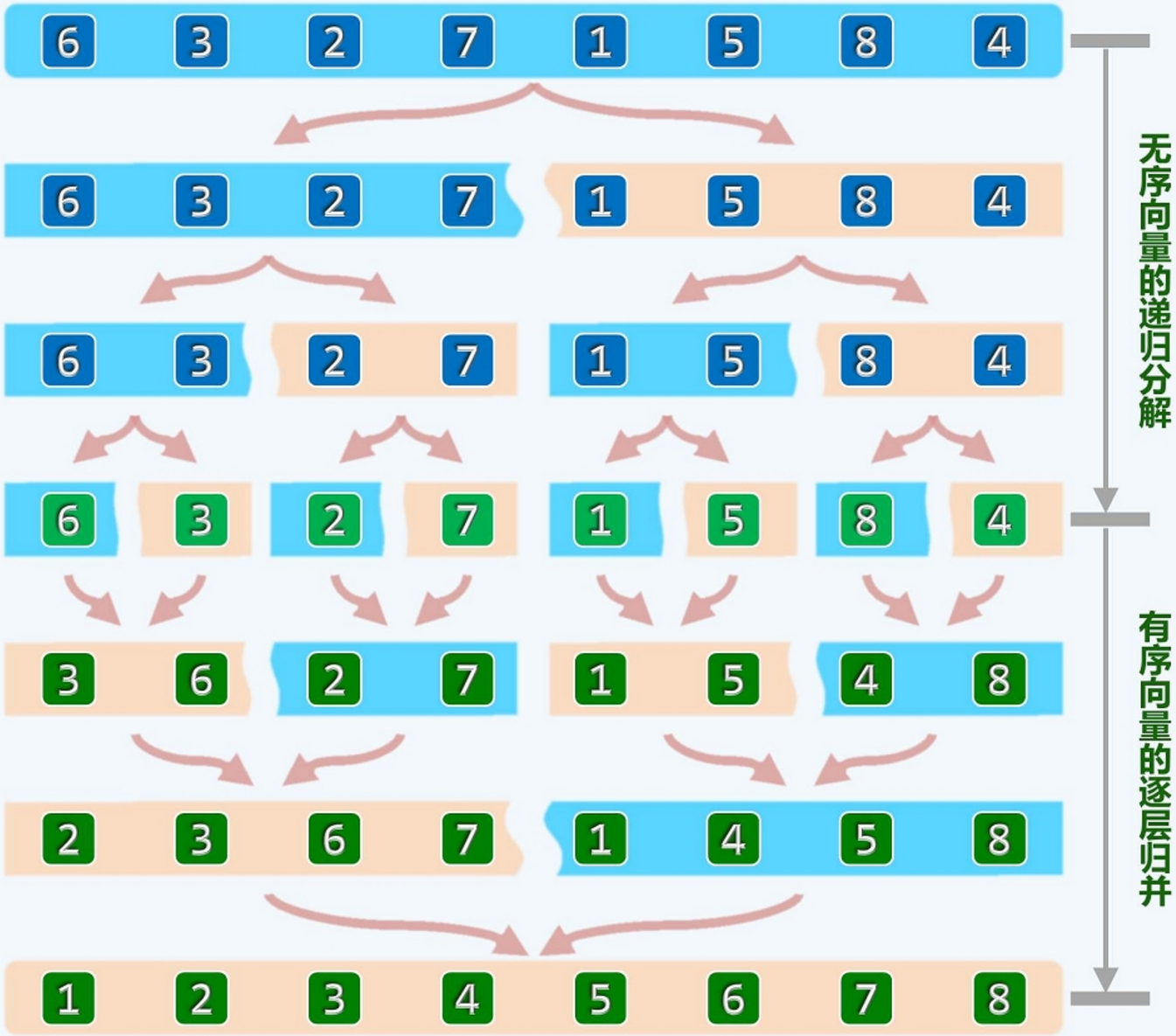
Mergesort

天下大势，分久必合，合久必分

Divide-And-Conquer

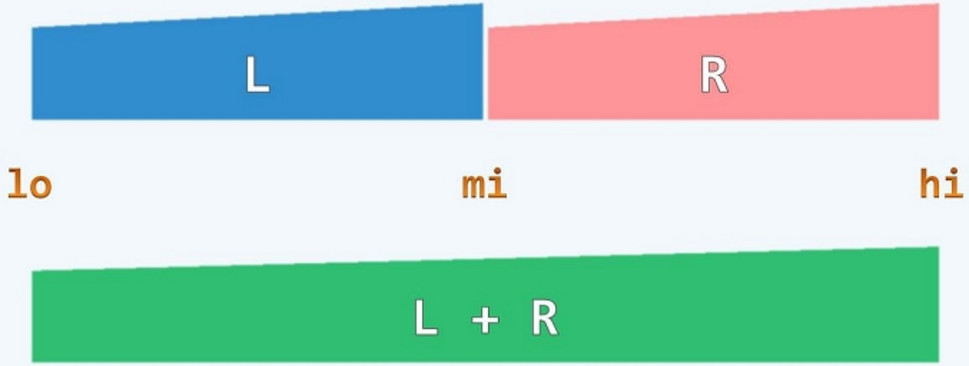
$$T(n) = 2 \cdot T(n/2) + O(n)$$

$$= O(n \cdot \log n)$$



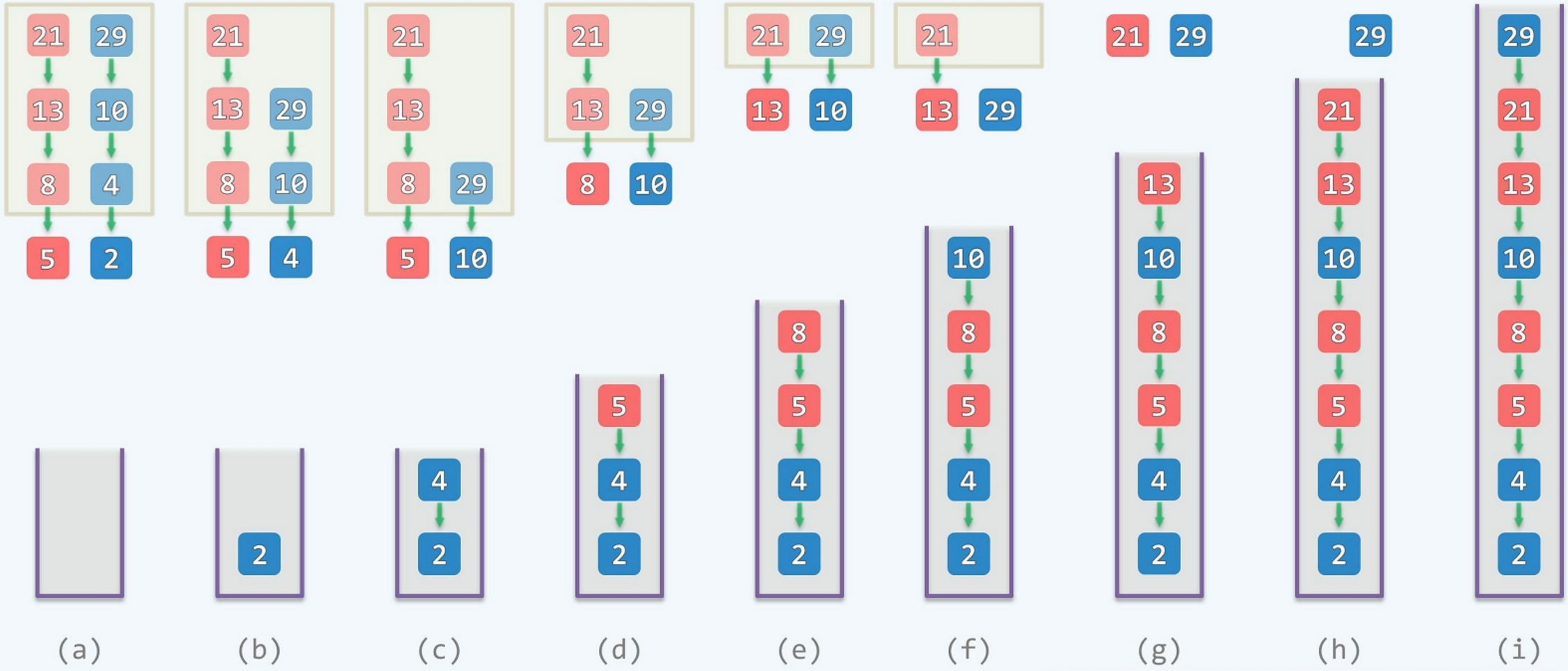
mergesort(lo, hi)

```
template <typename T>
void Vector<T>::mergeSort( Rank lo, Rank hi ) { //[lo, hi)
   if ( hi - lo < 2 ) return; //单元素区间自然有序，否则...
   int mi = (lo + hi) >> 1; //以中点为界
   mergeSort( lo, mi ); //对前半段排序
   mergeSort( mi, hi ); //对后半段排序
   merge( lo, mi, hi ); //归并
}
```



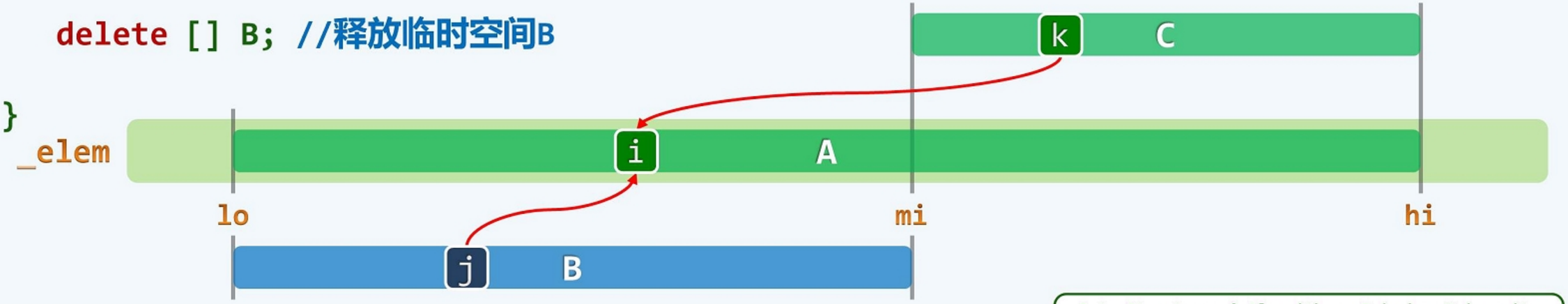
2-Way Merge

❖ S[lo, hi) = S[lo, mi) + S[mi, hi)



merge(lo, mi, hi)

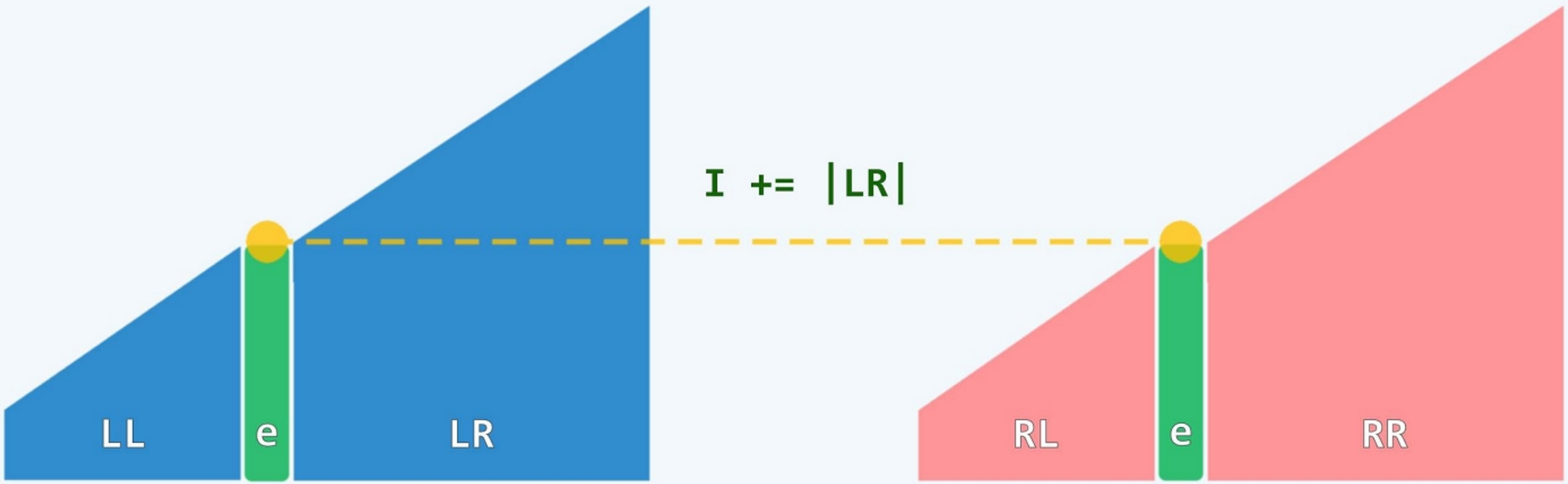
```
template <typename T> void Vector<T>::merge( Rank lo, Rank mi, Rank hi ) {
   T* A = _elem + lo; int lb = mi - lo; T* B = new T[lb]; //A[0, hi - lo) = _elem[lo, hi)
   for ( Rank i = 0; i < lb; i++ ) B[i] = A[i]; //复制前子向量B[0, lb) = _elem[lo, mi)
   int lc = hi - mi; T* C = _elem + mi; //后子向量C[0, lc) = _elem[mi, hi)
   for ( Rank i = 0, j = 0, k = 0; j < lb; ) //归并：反复从B和C首元素中取出更小者
      A[i++] = ( lc <= k || B[j] <= C[k] ) ? B[j++] : C[k++]; //将其归入A中
   delete [] B; //释放临时空间B
}
```



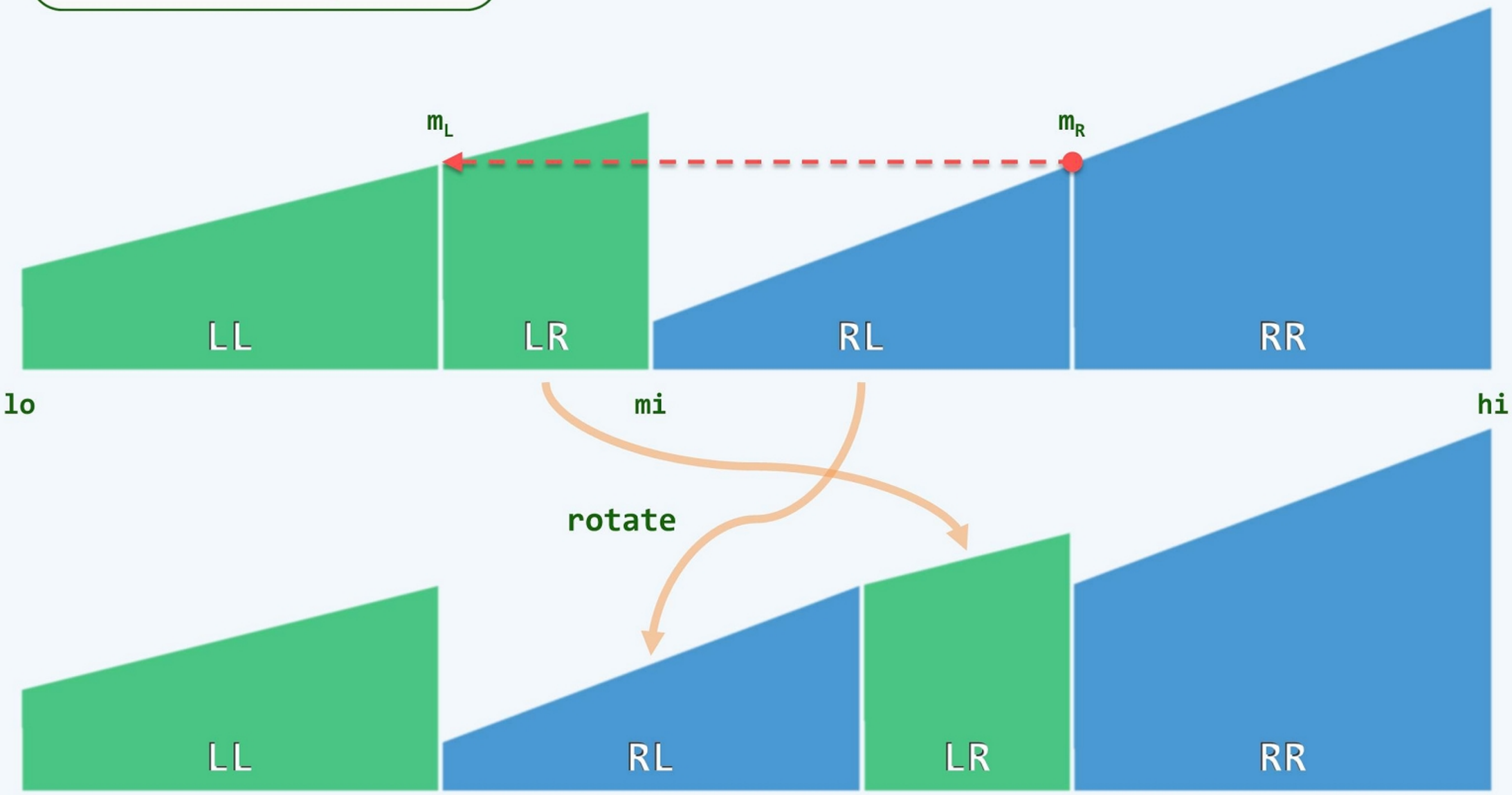
Counting Inversions

$\Omega(n^2)$

$\mathcal{O}(n \log n)$



C++ STL Mergesort



Shellsort

瓜熟蒂落，水到渠成

Example: $h_5 = 8$

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

Example: $h_4 = 5$

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

Example: $h_3 = 3$

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

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

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

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

Example: $h_2 = 2$


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

Example: $h_1 = 1$

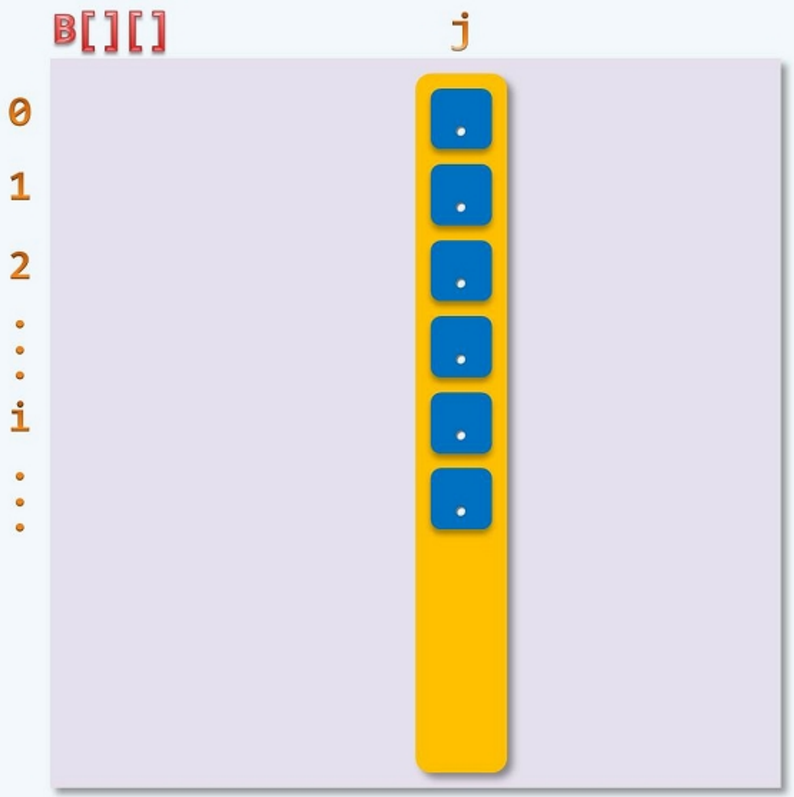
1 8 12 18 19 40 23 46 71 80 92 85 96

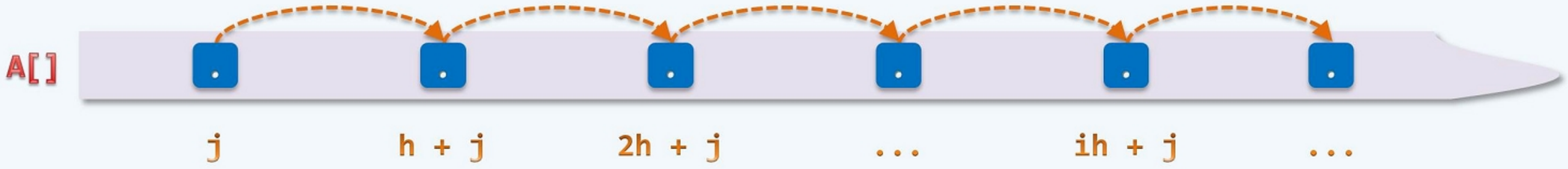
1 8 12 18 19 23 40 46 71 80 85 92 96

Call-By-Rank

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

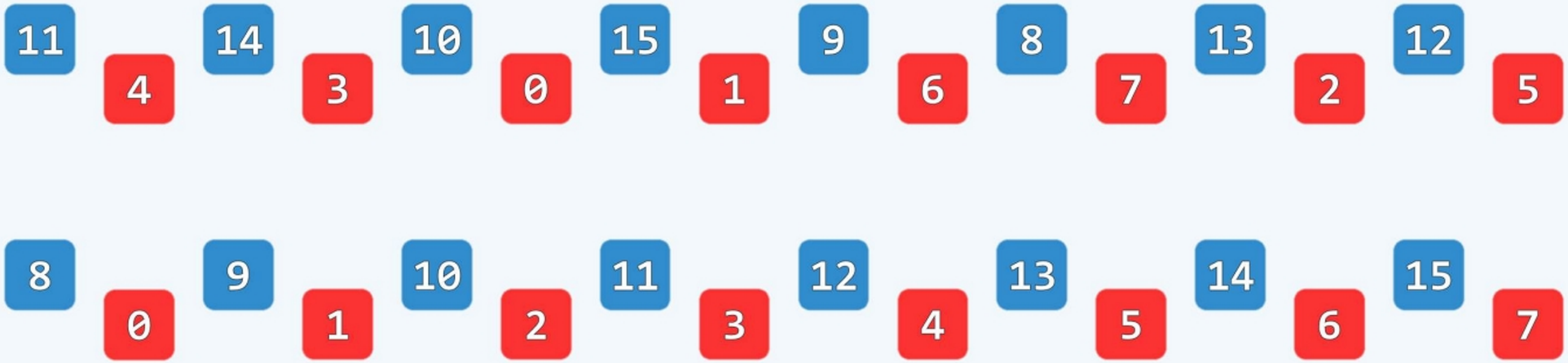
$$A[k] = B[k/h][k\%h]$$





Shell's Sequence

$$\mathcal{H}_{shell} = \{ 1, 2, 4, 8, \ldots, 2^k, \ldots \}$$



$$1 + 2 + 3 + \cdots + 2^{N-1} = \Omega(n^2)$$

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 $\boxed{\text{NOT}}$ 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

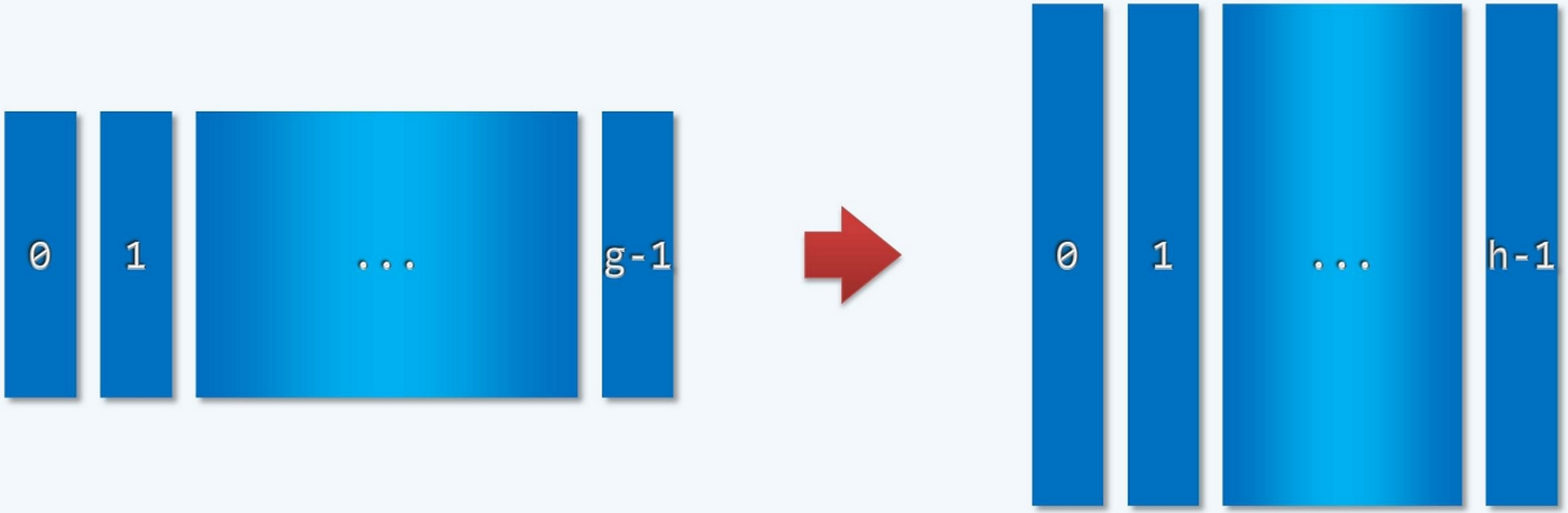
  $$\mathbf{x}(g,h) = (g-1) \cdot (h-1) - 1 = gh - g - h$$

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

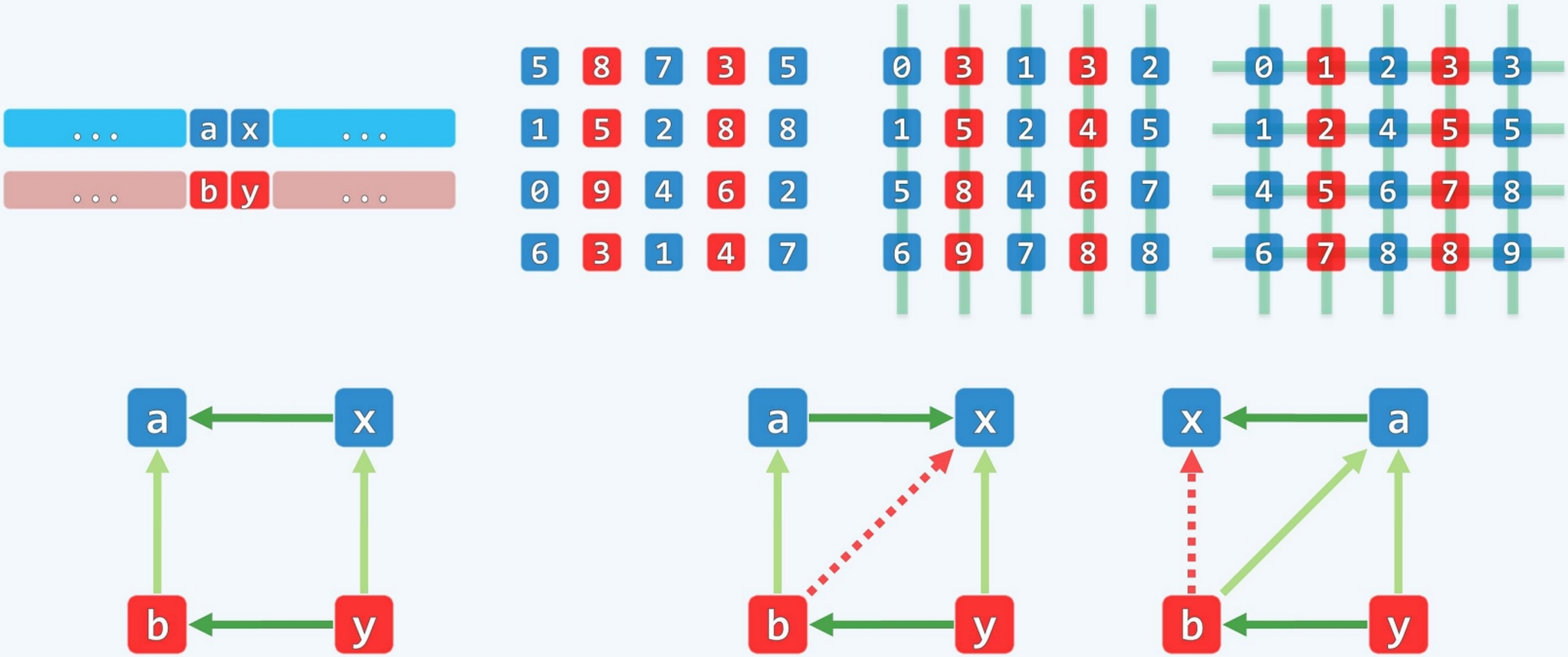
Theorem K

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

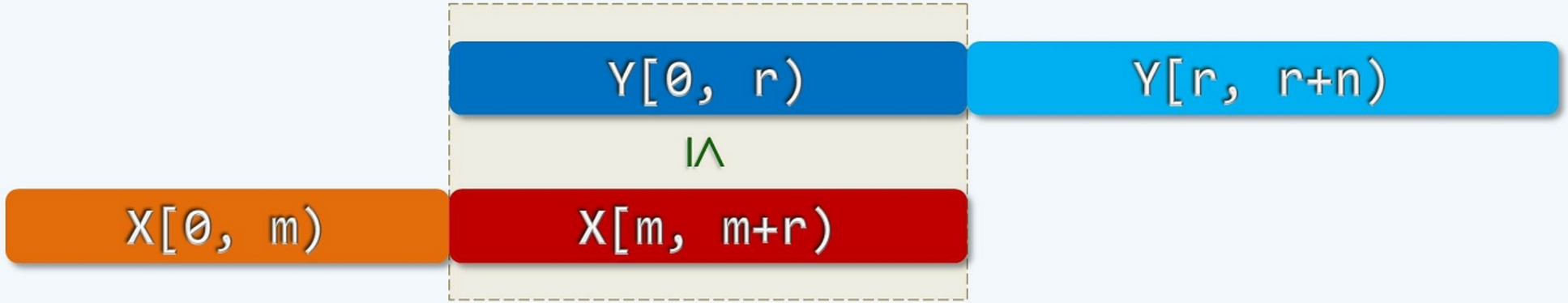
A g-ordered sequence REMAINS g-ordered after being h-sorted.

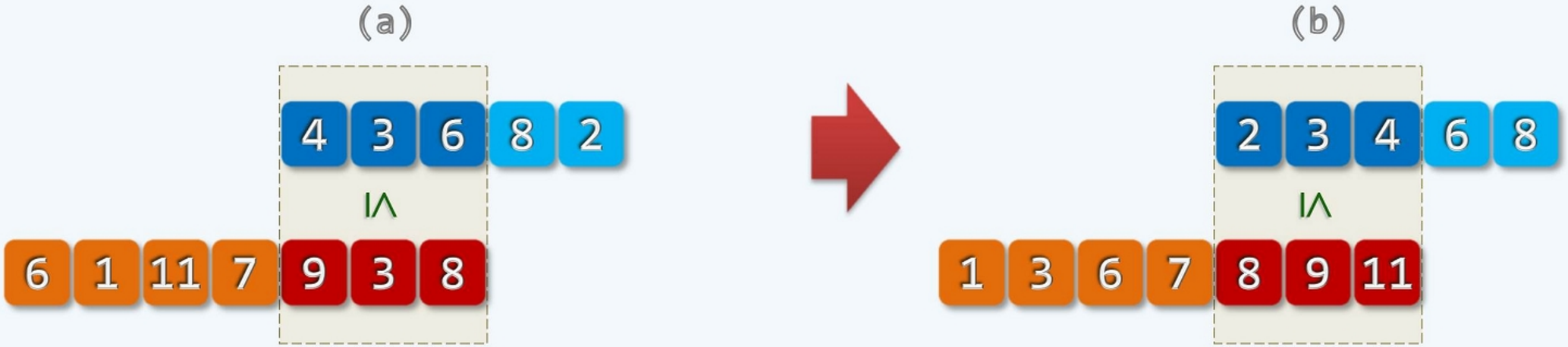


Order Preservation



Lemma L



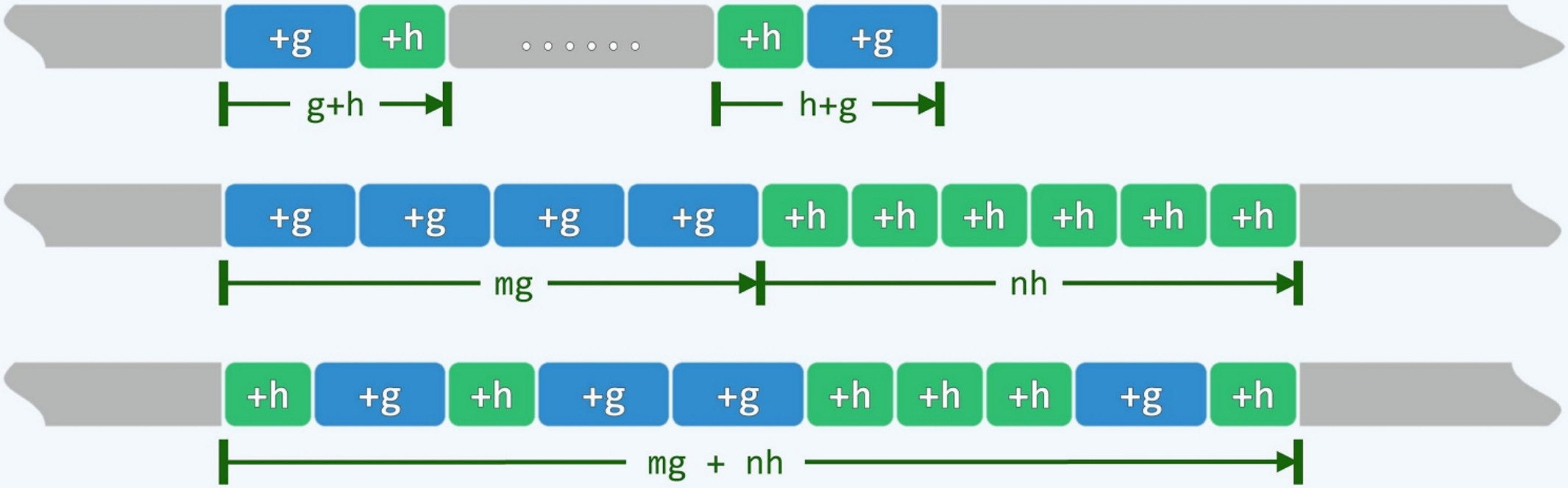


Linear Combination

❖ 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[i] and S[j], we have

$$j - i \;\geq\; \mathrm{x}(g,h) + 1 \;=\; (g-1)\cdot(h-1) \quad \text{only if} \quad S[i] \leq S[j]$$

- This implies that to the **RIGHT** of each element,

  only the next $\mathrm{x}(g,h)$ elements could be smaller

- There would be no more than $n \cdot \mathrm{x}(g,h)$ inversions altogether

PS Sequence

$$\{\ 1,\ 3,\ 7,\ 15,\ 31,\ 63,\ 127,\ 255,\ 511,\ 1023,\ \ldots\ \}$$

$$\mathcal{H}_{Shell}\ -\ 1\ =\ \{\ 2^k - 1 \mid k \in \mathcal{N}\ \}$$

$$\mathcal{O}(n^{3/2})\ =\ \mathcal{O}(n \cdot \sqrt{n})$$

Pratt Sequence

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

$$\{\ 2^p \cdot 3^q \ \mid\ p, q \in \mathcal{N}\ \}$$

$$\mathcal{O}(n \cdot \log^2 n)$$

Sedgewick Sequence

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

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

$$\mathcal{O}(n \cdot \log^2 n)$$