12.排序

(b2) 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 $\boxed{S_1}$ 和 $\boxed{S_2}$

如何快速找出有序向量 $\boxed{S = S_1 \cup S_2}$ 的中位数？

❖ 蛮力： 归并 $\boxed{S_1}$ 和 $\boxed{S_2}$ ，得到有序向量 $\boxed{S}$

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

❖ 如此，共需 $O(|S_1| + |S_2|)$ 时间

❖ 这一效率虽不算低，但毕竟未能充分利用 $\boxed{S_1}$ 和 $\boxed{S_2}$ 的 $\boxed{\text{有序性}}$

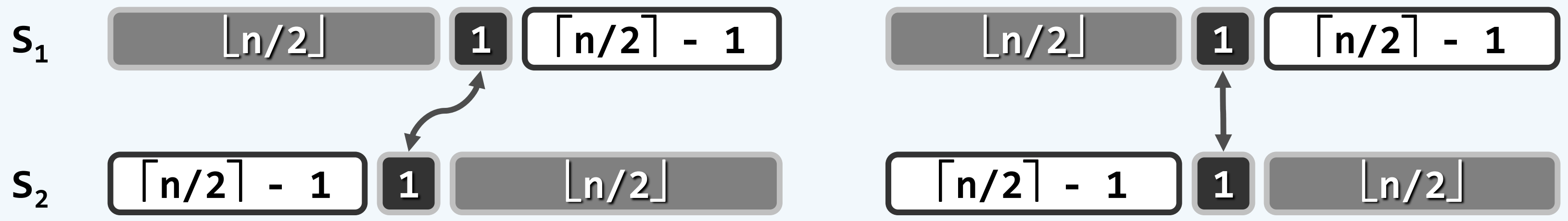
❖ 以下，先介绍 $\boxed{|S_1| = |S_2| = n}$ 情况下的算法

然后，再将该算法推广至一般情况

❖ 新的算法，依然采用 $\boxed{\text{减而治之}}$ 策略...

等长子向量：构思

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



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

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

或者$\boxed{\text{不是}}$S的中位数；或者与m_1或$m_2$$\boxed{\text{同为}}$S的中位数

这意味着，$\boxed{\text{剪除}}$这些区间之后，S中位数的数值$\boxed{\text{保持不变}}$

❖ 总之，每经一次比较，原问题的规模即$\boxed{\text{大致减半}}$——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 );
} //O( log( min( n1, n2 ) ) )
```

可见，实际上等长版本才是难度最大的