

2. 向量

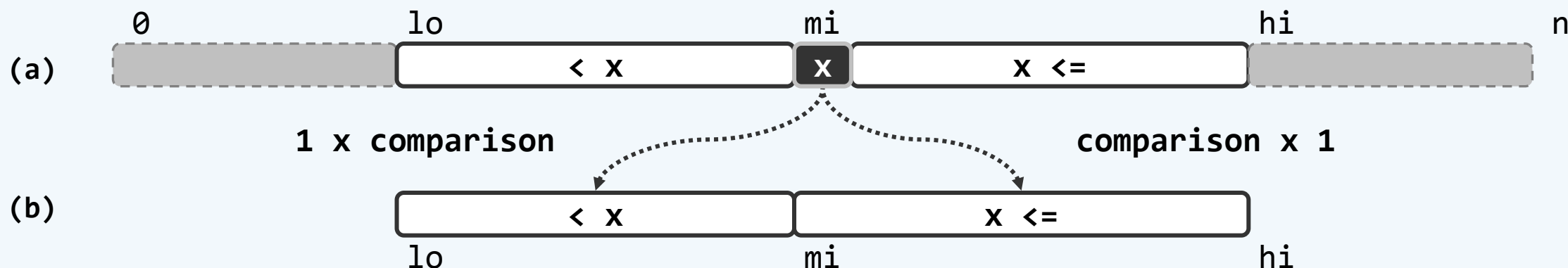
(d4) 有序向量：二分查找（改进）

邓俊辉

deng@tsinghua.edu.cn

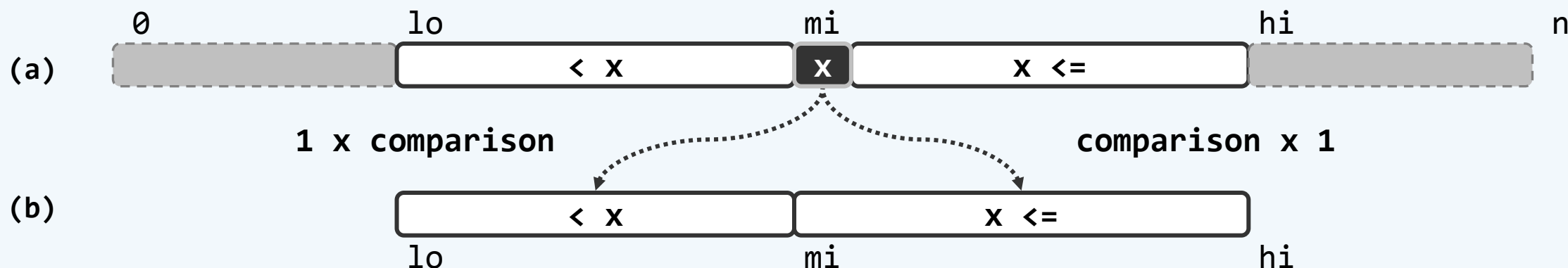
版本B：改进思路

- ❖ 二分查找中左、右分支转向代价不平衡的问题，也可直接解决
- ❖ 比如，每次迭代（或每个递归实例）仅做1次关键码比较
如此，所有分支只有2个方向，而不再是3个
- ❖ 同样地，轴点 mi 取作中点，则查找每深入一层，问题规模也缩减一半
 - 1) $e < x$ ：则 e 若存在必属于左侧子区间 $s[lo, mi)$ ，故可递归深入
 - 2) $x \leq e$ ：则 e 若存在必属于右侧子区间 $s[mi, hi)$ ，亦可递归深入只有当元素数目 $hi - lo = 1$ 时，才判断该元素是否命中



版本B：实现

```
❖ template <typename T> static Rank binSearch(T* A, T const & e, Rank lo, Rank hi) {  
    while (1 < hi - lo) { //有效查找区间的宽度缩短至1时，算法才会终止  
        Rank mi = (lo + hi) >> 1; //以中点为轴点，经比较后确定深入  
        (e < A[mi]) ? hi = mi : lo = mi; //[lo, mi)或[mi, hi)  
    } //出口时hi = lo + 1, 查找区间仅含一个元素A[lo]  
    return (e == A[lo]) ? lo : -1 ; //返回命中元素的秩或者-1  
} //相对于版本A，最好（坏）情况下更坏（好）；各种情况下的SL更加接近，整体性能更趋稳定
```



版本B：语义约定

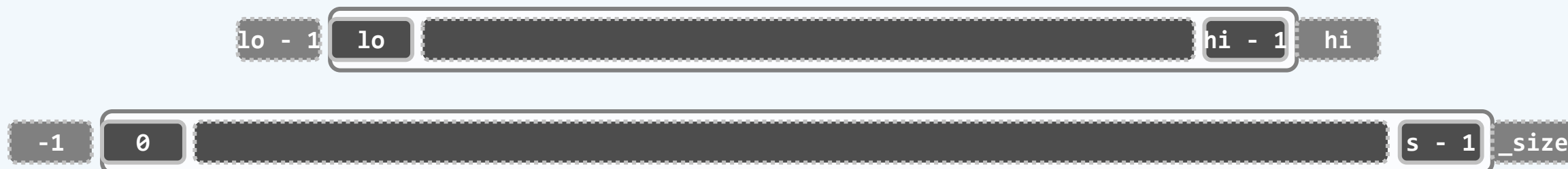
❖ 以上二分查找及Fibonacci查找算法

均未严格地兑现search()接口的语义约定：返回不大于e的最后一个元素

❖ 只有兑现这一约定，才可有效支持相关算法，比如： $V.\text{insert}(1 + V.\text{search}(e), e)$

1) 当有多个命中元素时，必须返回最靠后（秩最大）者

2) 失败时，应返回小于e的最大者（含哨兵[lo - 1]）



版本C：实现

```
❖ template <typename T> static Rank binSearch(T* A, T const& e, Rank lo, Rank hi) {  
    while (lo < hi) { //不变性：A[0, lo) <= e < A[hi, n)  
        Rank mi = (lo + hi) >> 1; //以中点为轴点，经比较后确定深入  
        (e < A[mi]) ? hi = mi : lo = mi + 1; //[lo, mi)或(mi, hi)  
    } //出口时，A[lo = hi]为大于e的最小元素  
    return --lo; //故lo - 1即不大于e的元素的最大秩  
}
```

❖ 与版本B的差异

- 1) 待查找区间宽度缩短至 0 而非 1 时，算法才结束
- 2) 转入右侧子向量时，左边界取作 $mi + 1$ 而非 mi — $A[mi]$ 会被遗漏？
- 3) 无论成功与否，返回的秩严格符合接口的语义约定...

版本C：正确性

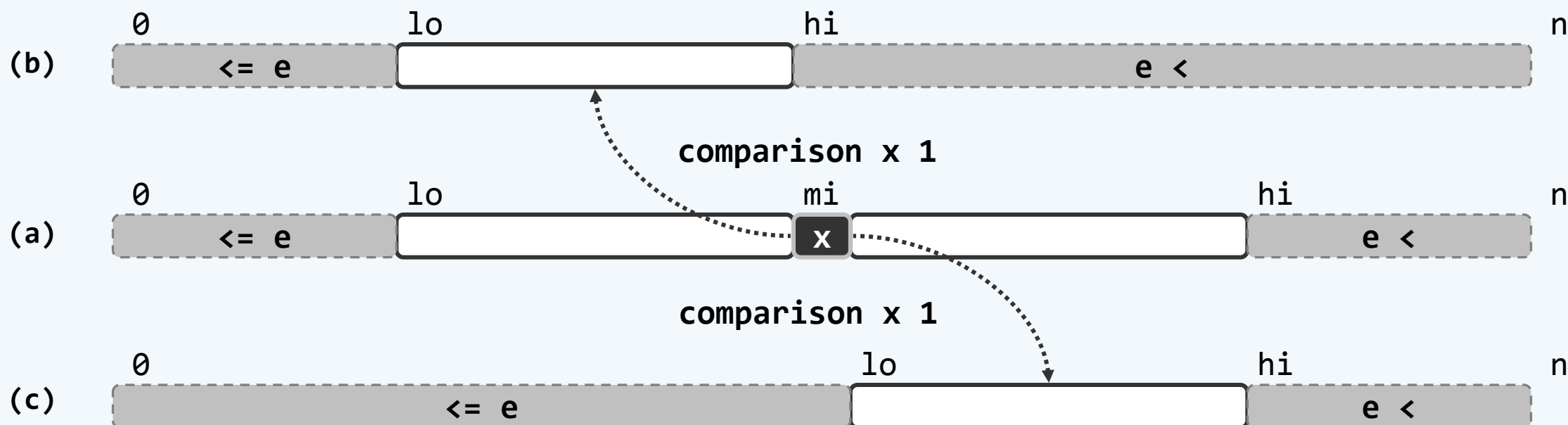
❖ 不变性： $A[lo, hi) \leq e < A[hi, n)$

// $A[hi]$ 总是大于 e 的最小者

❖ 初始时， $lo = 0$ 且 $hi = n$ ， $A[lo, lo) = A[hi, n) = \emptyset$ ，自然成立

❖ 数学归纳：假设不变性一直保持至 (a)，以下无非两种情况...

❖ 单调性：显而易见



课后

❖ 针对二分查找，Knuth (ACP-v3-s6.2.1-ex_23) 曾指出：

将三分支变为两分支后的改进效果

需要到n非常大 ($2^{2*(17-(-16))} = 2^{66}$) 后方能体现

试阅读相关段落；这一结论，对当下的实际应用有何意义？