

## 2. 向量

### (d2) 有序向量：二分查找

群猴道：“自从爷爷去后，这山被二郎菩萨点上火，烧杀了大半。我们蹲在井里，钻在涧内，藏于铁板桥下，得了性命。及至火灭烟消，出来时，又没花果养赡，难以存活，别处又去了一半。我们这一半，捱苦的住在山中，这两年，又被些打猎的抢了一半去也。”

邓俊辉

deng@tsinghua.edu.cn

## 统一接口

```
❖ template <typename T> //查找算法统一接口,  $0 \leq lo < hi \leq \_size$   
    Rank Vector<T>::search(T const & e, Rank lo, Rank hi) const {  
        return (rand() % 2) ? //按各50%的概率随机选用  
            binSearch(_elem, e, lo, hi) //二分查找算法, 或者  
            : fibSearch(_elem, e, lo, hi); //Fibonacci查找算法  
    }
```



❖ 如何处置特殊情况？

比如，目标元素不存在；或者反过来，目标元素同时存在多个

## 语义约定

❖ 至少，应该便于有序向量自身的维护： $V.\text{insert}(1 + V.\text{search}(e), e)$

即便失败，也应给出新元素适当的插入位置

若允许重复元素，则每一组也需按其插入的次序排列



❖ 约定：在有序向量区间  $V[lo, hi)$  中，确定 **不大于  $e$  的最后一个元素**

若  $-\infty < e < V[lo]$ ，则 返回  $lo - 1$  (左侧哨兵)

若  $V[hi - 1] < e < +\infty$ ，则 返回  $hi - 1$  (末元素 = 右侧哨兵左邻)

## 版本A：原理

❖ 减而治之：以任一元素  $x = S[mi]$  为界，都可将待查找区间分为三部分

$$S[lo, mi) \leq S[mi] \leq S(mi, hi)$$

//  $S[mi]$  称作轴点

❖ 只需将目标元素  $e$  与  $x$  做一比较，即可分三种情况进一步处理：

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

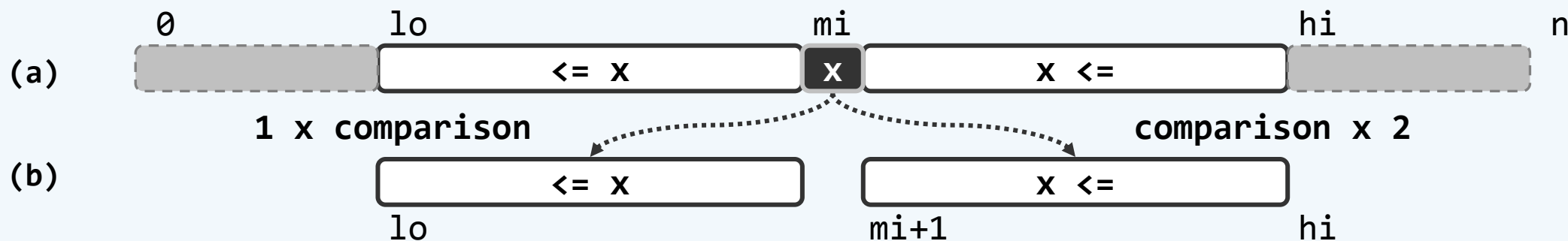
2)  $x < e$ ：则  $e$  若存在必属于右侧子区间  $S(mi, hi)$ ，亦可递归深入

3)  $e = x$ ：已在此处命中，可随即返回

// 若有多个，返回何者？

❖ 二分（折半）策略：轴点  $mi$  总是取作中点——于是

每经过至多两次比较，或者能够命中，或者将问题规模缩减一半



## 版本A：实现

❖ template <typename T> //在有序向量区间[lo, hi)内查找元素e

```
static Rank binSearch(T* A, T const& e, Rank lo, Rank hi) {
```

```
    while (lo < hi) { //每步迭代可能要做两次比较判断，有三个分支
```

```
        Rank mi = (lo + hi) >> 1; //以中点为轴点
```

```
        if      (e < A[mi]) hi = mi; //深入前半段[lo, mi)继续查找
```

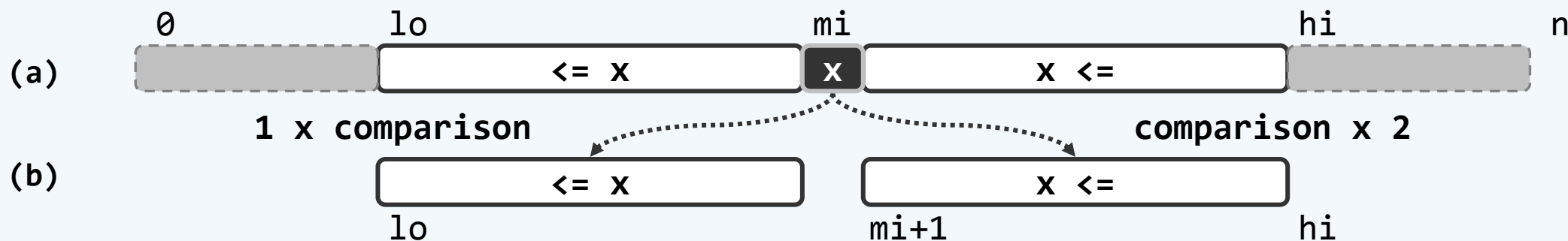
```
        else if (A[mi] < e) lo = mi + 1; //深入后半段(mi, hi)
```

```
        else          return mi; //在mi处命中
```

```
    }
```

```
    return -1; //查找失败
```

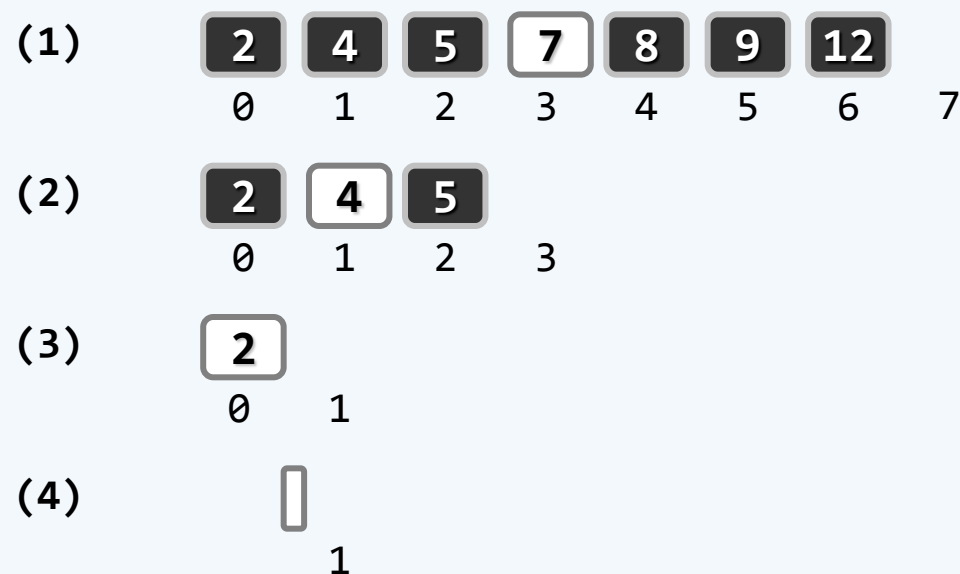
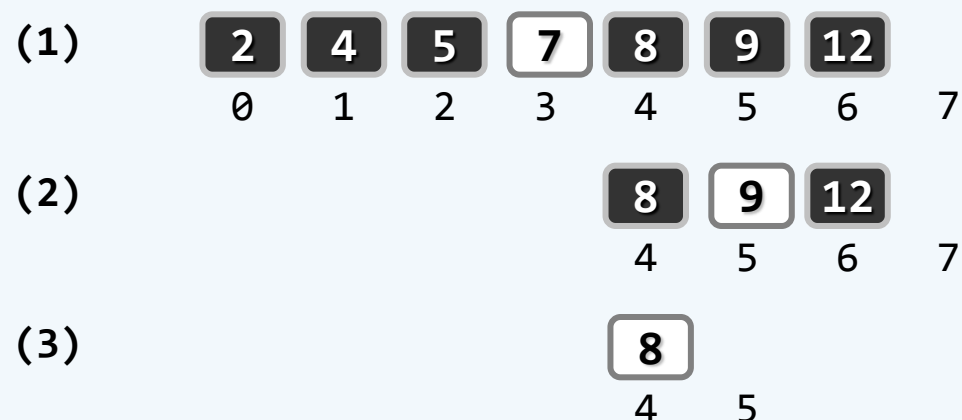
```
}
```



## 版本A：实例与复杂度

❖  $S.search(8, 0, 7)$  : 共经  $2 + 1 + 2 = 5$  次比较, 在  $S[4]$  处命中

❖  $S.search(3, 0, 7)$  : 共经  $1 + 1 + 2 = 4$  次比较, 在  $S[1]$  处失败



❖ 线性递归： $T(n) = T(n/2) + O(1) = O(\log n)$ ，大大优于顺序查找

递归跟踪：轴点总取中点，递归深度 $O(\log n)$ ；各递归实例均耗时 $O(1)$

## 版本A：查找长度

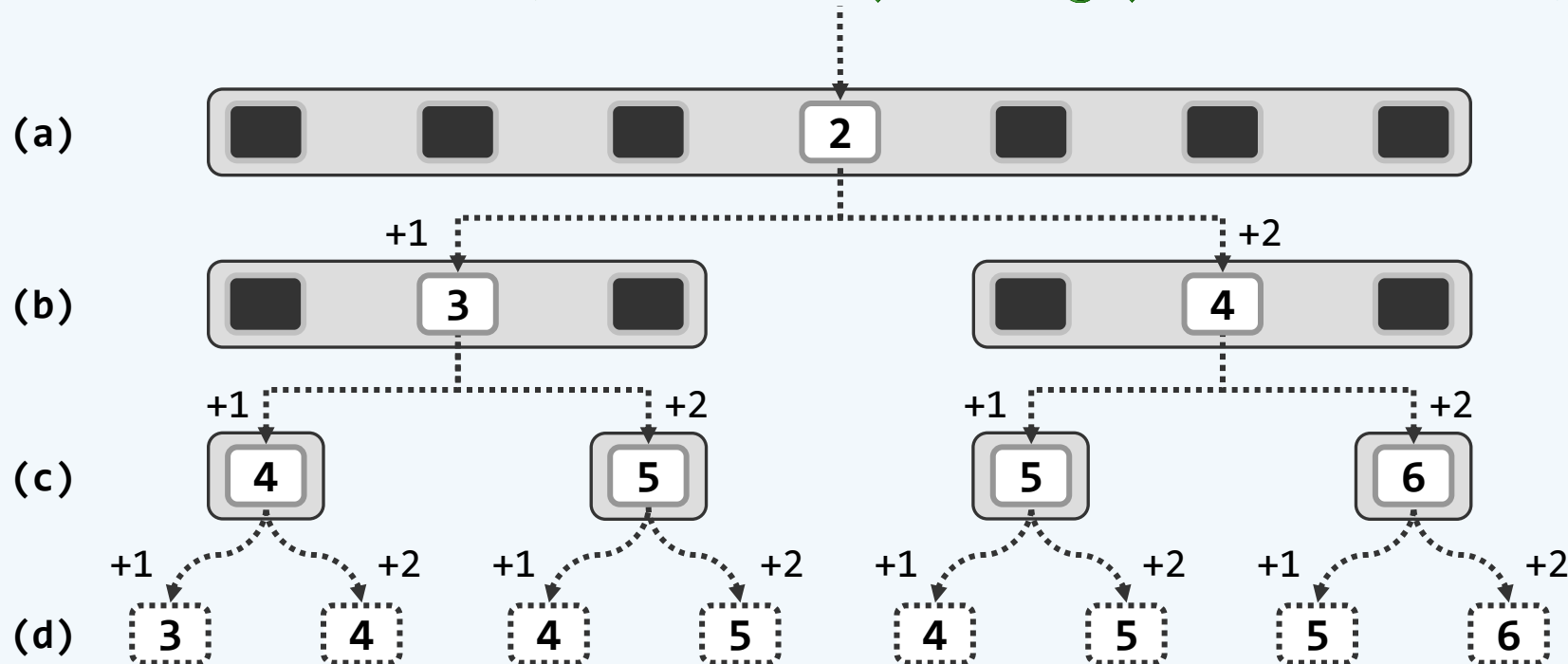
❖ 如何更为精细地评估查找算法的性能？

考查关键码的比较次数，即查找长度（search length）

❖ 通常，需分别针对成功与失败查找，从最好、最坏、平均等角度评估

❖ 比如，成功、失败时的平均查找长度均大致为 $O(1.50 \cdot \log n)$

//详见教材、习题解析



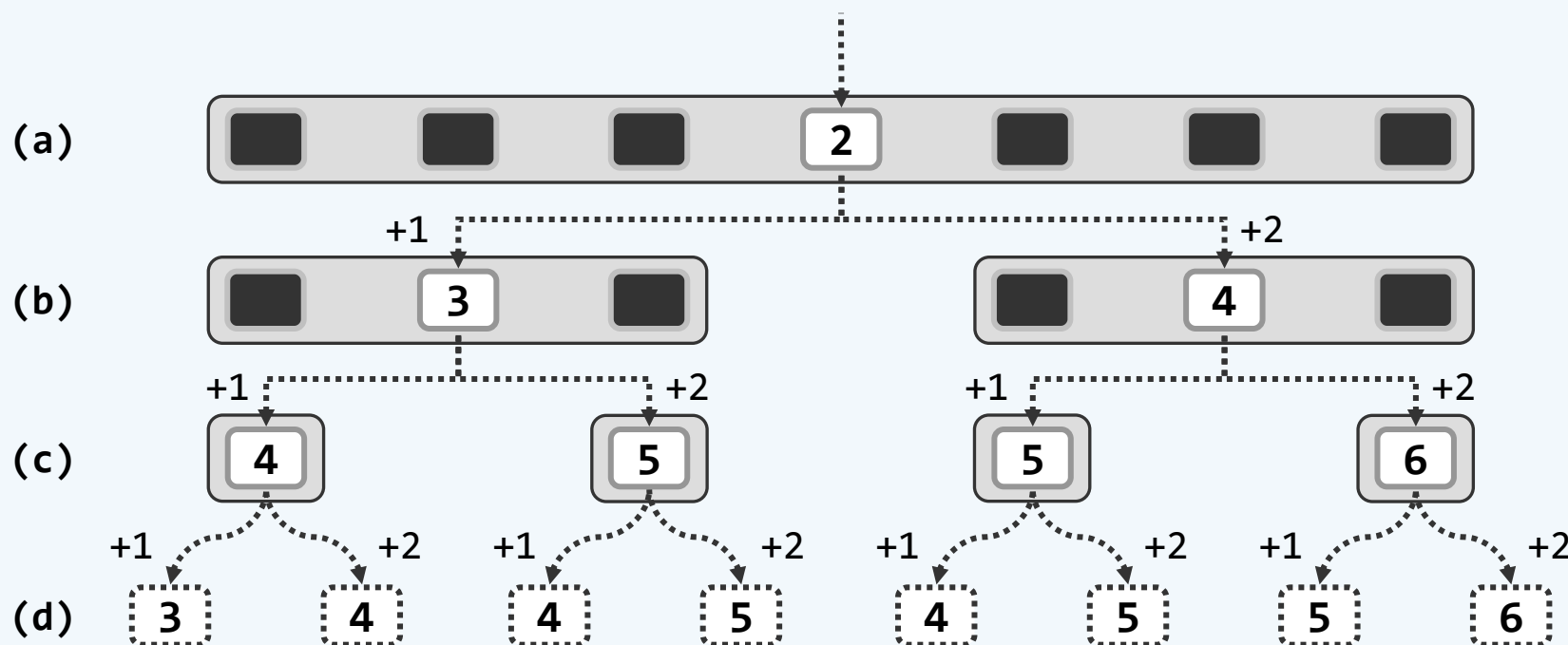
## 版本A：查找长度

❖  $n = 7$ 时，各元素对应的成功查找长度为{ 4, 3, 5, 2, 5, 4, 6 }

在等概率情况下，平均**成功**查找长度 =  $29 / 7 = 4.14$

❖ 共8种失败情况，查找长度分别为{3, 4, 4, 5, 4, 5, 5, 6}

在等概率情况下，平均**失败**查找长度 =  $36 / 8 = 4.50$





## 课后

❖ 各种查找结果出现的概率不均等时，查找长度应该如何定义和计算？