2.向量

有序向量

二分查找（版本A）

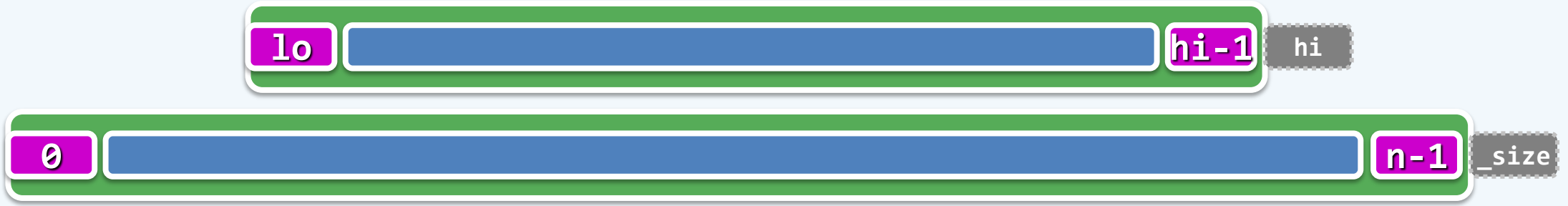
自从爷爷去后，这山被二郎菩萨点上火，烧杀了大半。我们蹲在井里，钻在涧内，藏于铁板桥下，得了性命。及至火灭烟消，出来时，又没花果养赡，难以存活，别处又去了一半。我们这一半，捱苦的住在山中，这两年，又被些打猎的抢了一半去也。

邓俊辉

deng@tsinghua.edu.cn

统一接口

```
❖ template <typename T> //查找算法统一接口，0 <= lo < hi <= _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
  }
```



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[mi]称作轴点

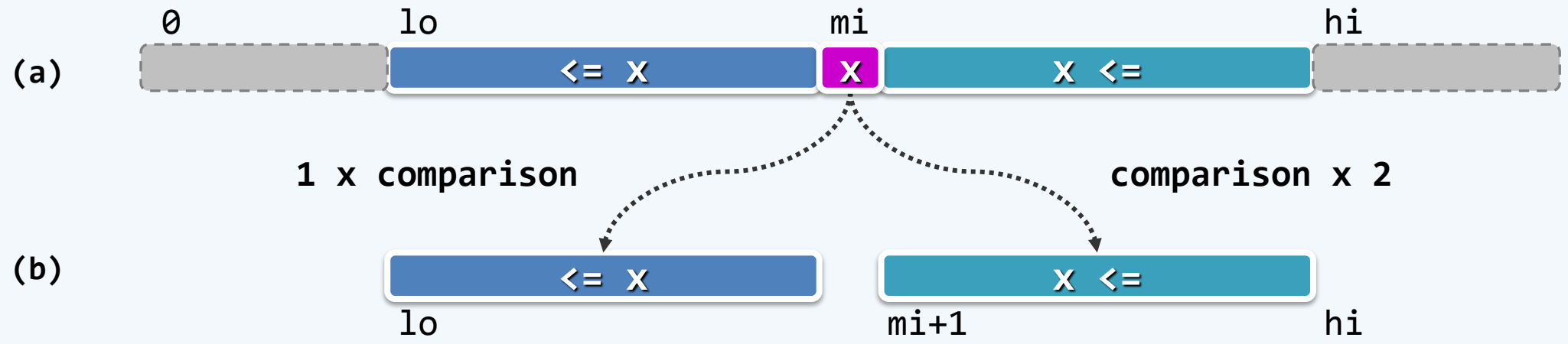
❖ 既然向量整体有序，则必有：

S[lo, mi) <= S[mi] <= S(mi, hi)

减而治之

❖ 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：

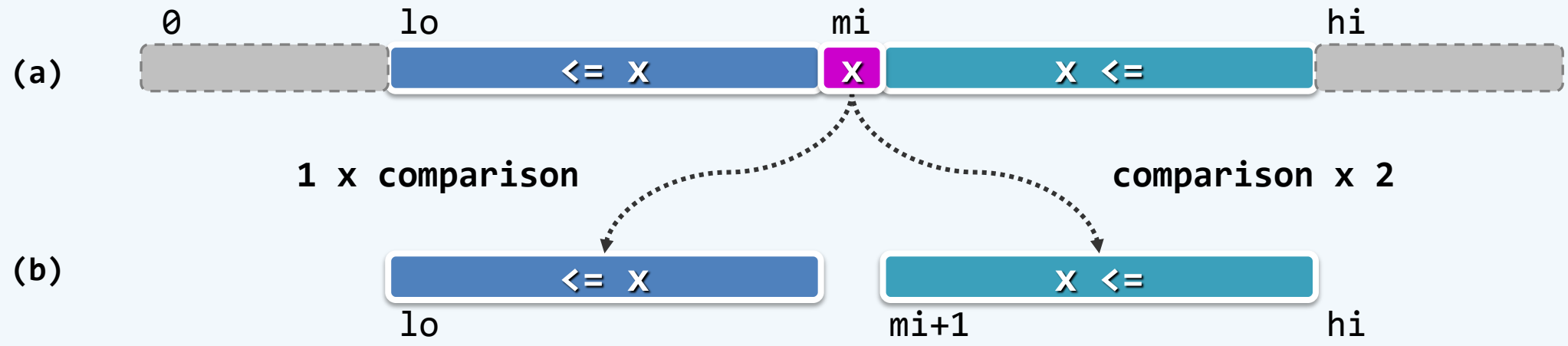
- e < x：则e若存在必属于左侧子区间S[lo, mi)，故可递归深入
- x < e：则e若存在必属于右侧子区间S(mi, hi)，亦可递归深入
- e = x：已在此处命中，可随即返回 //若有多个，返回何者？



二分折半

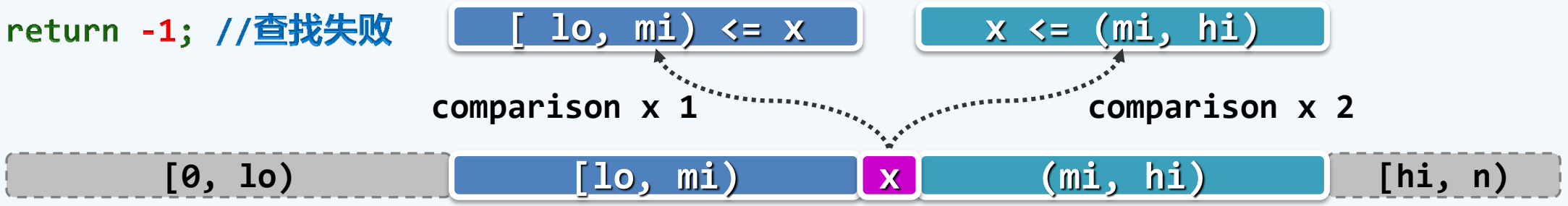
❖ 若轴点mi取作中点，则每经过至多两次比较

- 或者能够命中
- 或者将问题规模缩减一半



实现

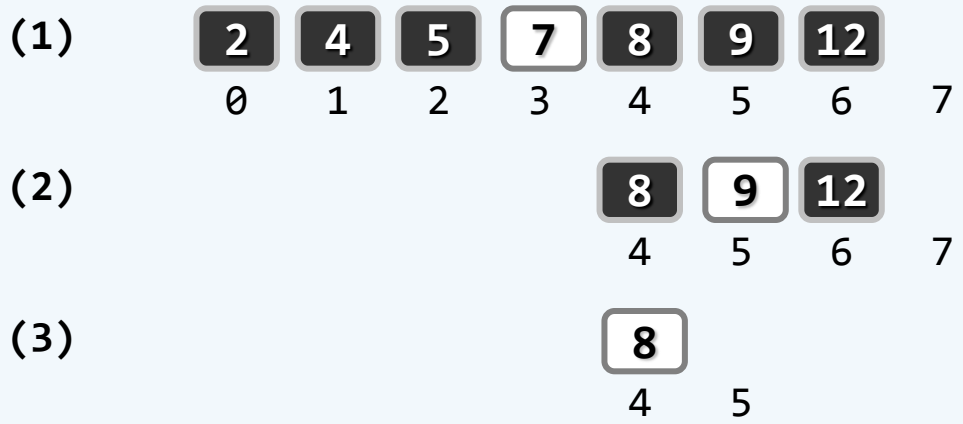
```
❖ template <typename T> //在有序向量区间[lo, hi)内查找元素e
  static Rank binSearch( T * A, T const & e, Rank lo, Rank hi ) {
     while ( lo < hi ) { //每步迭代可能要做两次比较判断，有三个分支
        Rank mi = ( lo + hi ) >> 1; //以中点为轴点
        if      ( e < A[mi] ) hi = mi; //深入前半段[lo, mi)继续查找
        else if ( A[mi] < e ) lo = mi + 1; //深入后半段(mi, hi)
        else                  return mi; //在mi处命中
     }
     return -1; //查找失败
  }
```

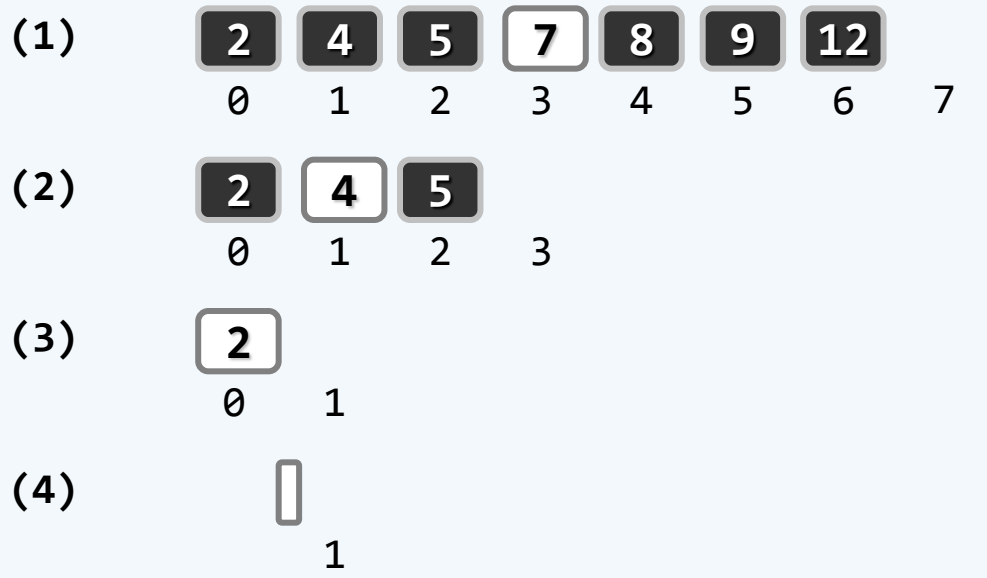


实例 + 复杂度

❖ 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(logn)，大大优于顺序查找

递归跟踪：轴点总取中点，递归深度O(logn)；各递归实例均耗时O(1)

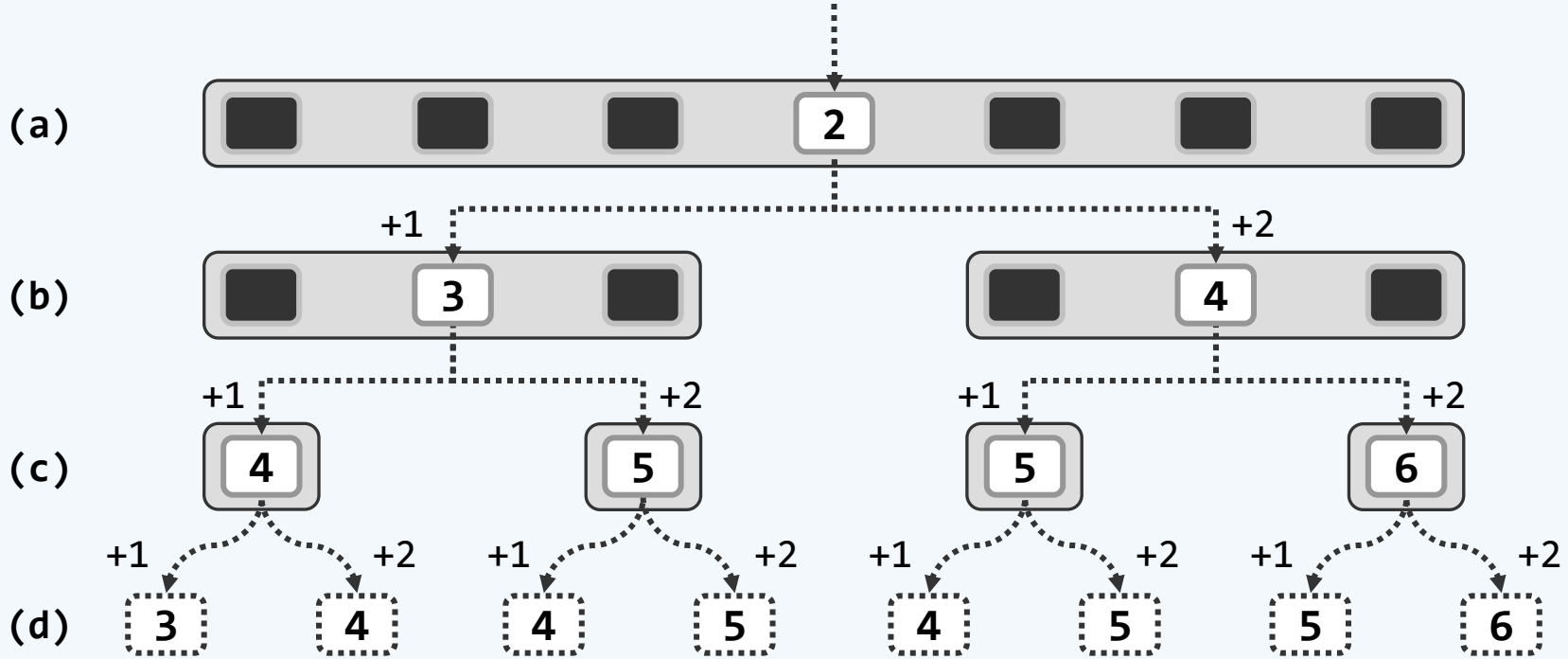
查找长度

- 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



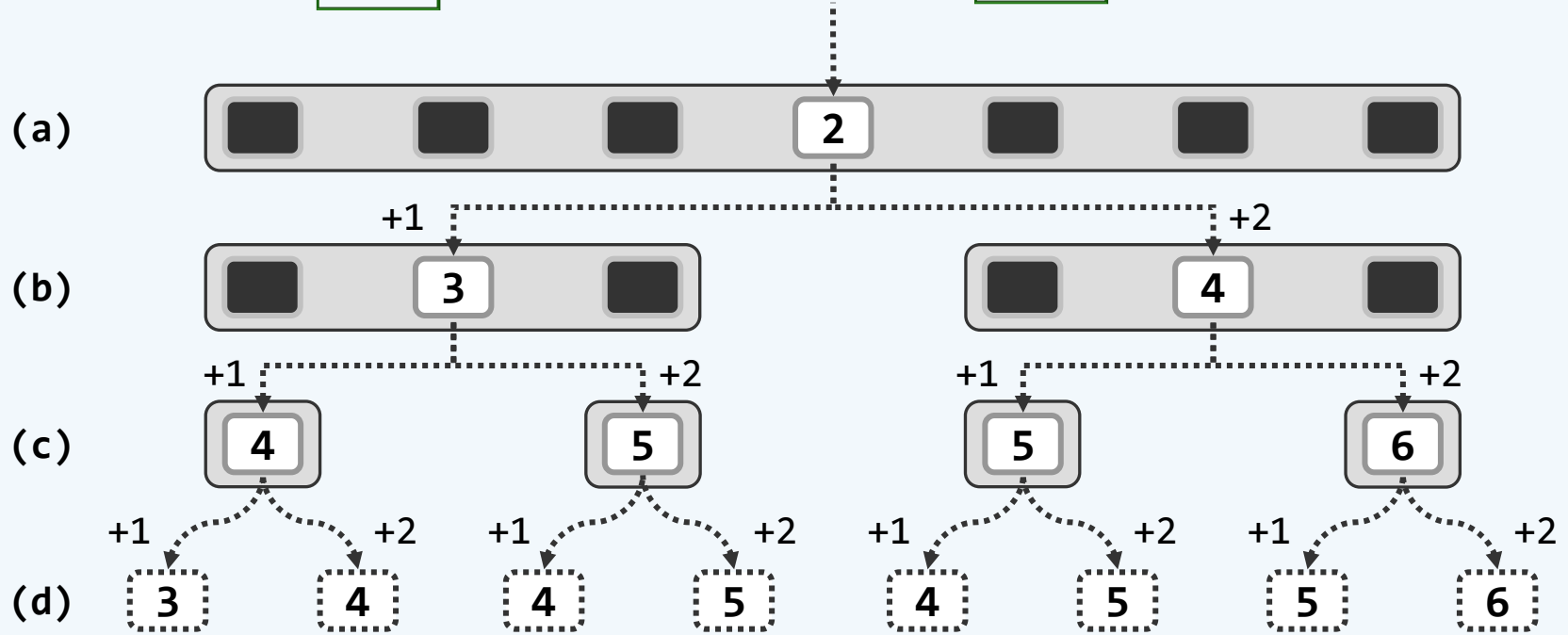
查找长度

❖ 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？