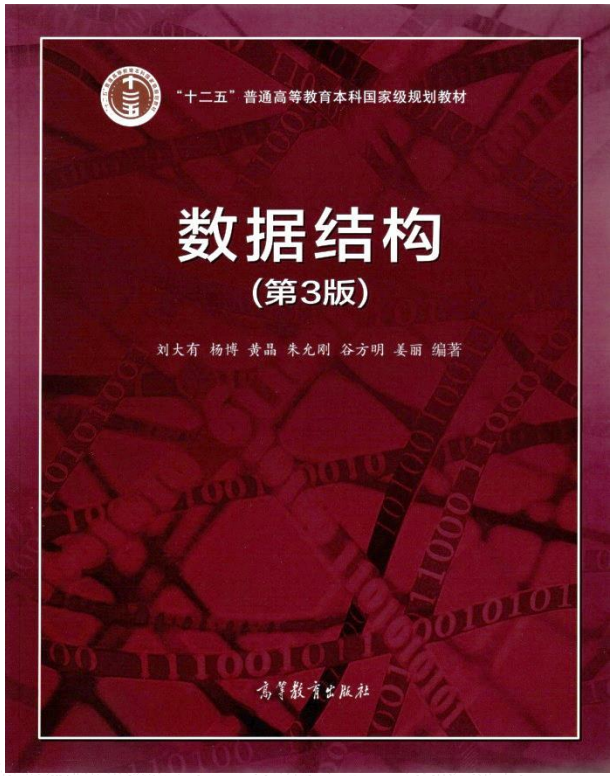
计算机学院王湘浩班

2024级



数据之法

结构之美

算法之道

线性结构查找

顺序查找

对半查找

插值查找

再谈对半查找

zhuyungang@jlu.edu.cn

Last updated on 2025.6

PSYCHOLOGY AND PHILOSOPHY

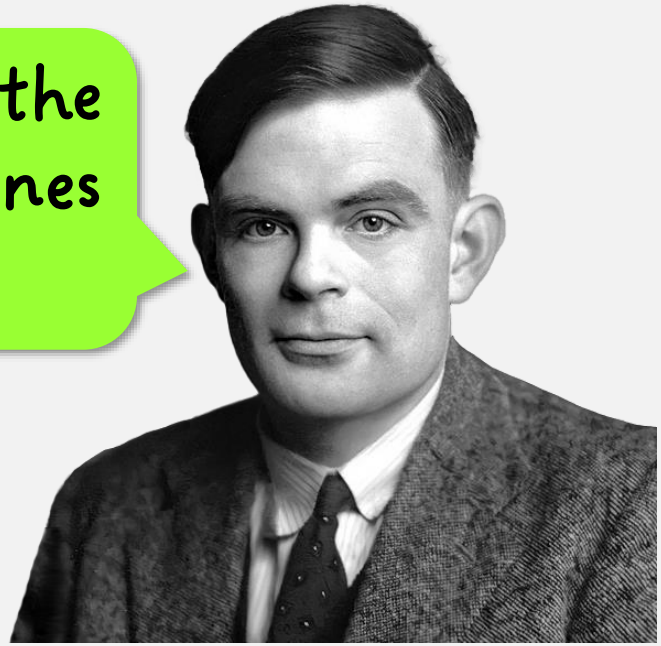
I.—COMPUTING MACHINERY AND INTELLIGENCE

By A. M. Turing

1. *The Imitation Game.*

I propose to consider the question, 'Can machines think?' This should begin with definitions of the meaning of the terms 'machine' and 'think'. The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous. If the meaning of the words 'machine' and 'think' are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning

I propose to consider the question "Can machines think?"



Alan Turing

The question of whether computers can think is like the question of whether submarines can swim.

Edsger Dijkstra

查找的基本概念

A

- 定义：查找亦称检索。给定一个文件包含n个记录（或称元素、结点），每个记录都有一个关键词域。一个查找算法，就是对给定的值K，在文件中找关键词等于K的那个记录。
- 查找结果：成功、失败。
- 平均查找长度：查找一个元素所作的关键词平均比较次数，是衡量一个查找算法优劣的主要标准。

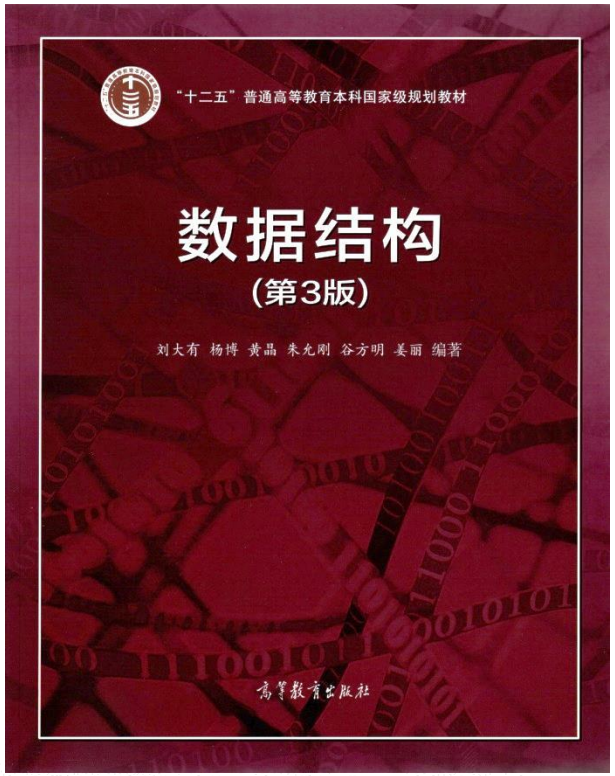
无序表的顺序查找

A

在线性表$R_1,R_2,\ldots,R_n$中查找关键词等于K的元素：从线性表的起始元素开始，逐个检查每个元素R_i $(1\le i \le n)$，若查找成功，返回K在R中的下标，若查找失败，返回-1。

```
int Search(int R[], int n, int K){
    for(int i=1; i<=n; i++)
        if(R[i]==K) return i;
    return -1;
}
```

时间复杂度
$O(n)$



线性结构查找

顺序查找

对半查找

插值查找

再谈对半查找

数据之法

结构之美

算法之道

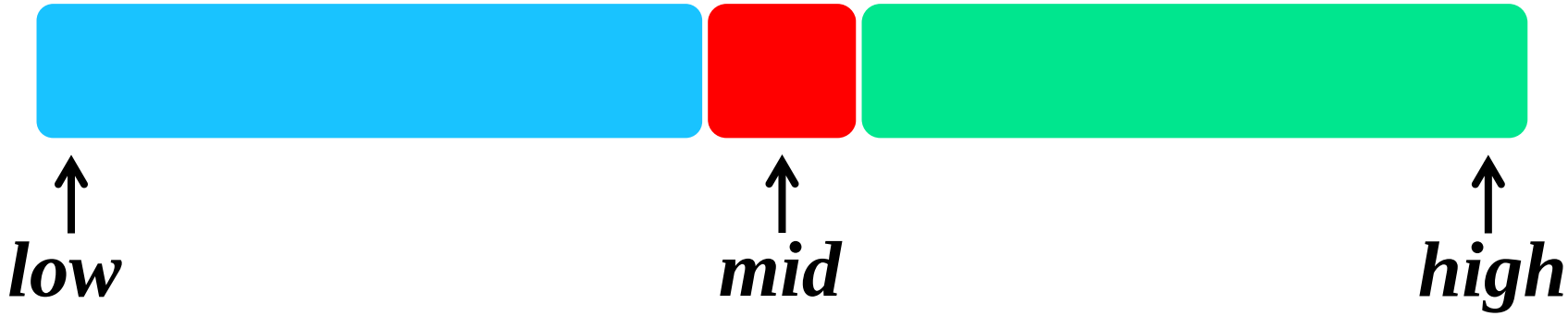
zhuyungang@jlu.edu.cn

Last updated on 2025.6

有序表的二分查找

A

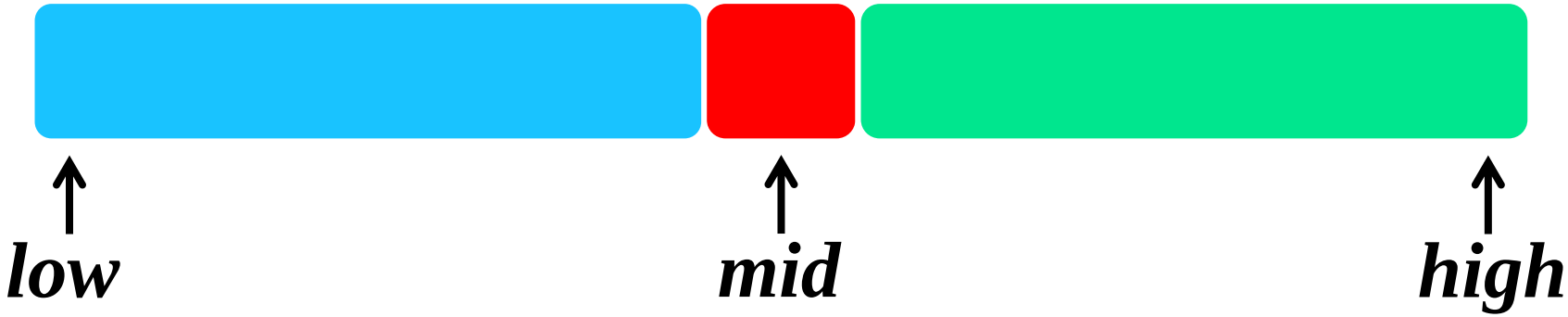
- 有序表$R_{low}, R_{low+1}, \ldots, R_{high}$按照关键词递增有序。
- 选取一个位置mid $(low \le mid \le high)$，比较K和R_{mid}，若：
  - ✓$K < R_{mid}$，[K只可能在R_{mid}左侧]
  - ✓$K > R_{mid}$，[K只可能在R_{mid}右侧]
  - ✓$K = R_{mid}$，[查找成功结束]
- 使用不同的规则确定mid，可得到不同的二分查找方法：对半查找、一致对半查找、斐波那契查找、插值查找等。



对半（折半）查找

A

- *K*与待查表的中间记录进行比较，即$mid \leftarrow \lfloor (low+high)/2 \rfloor$
- 每次迭代可将查找范围缩小一半。

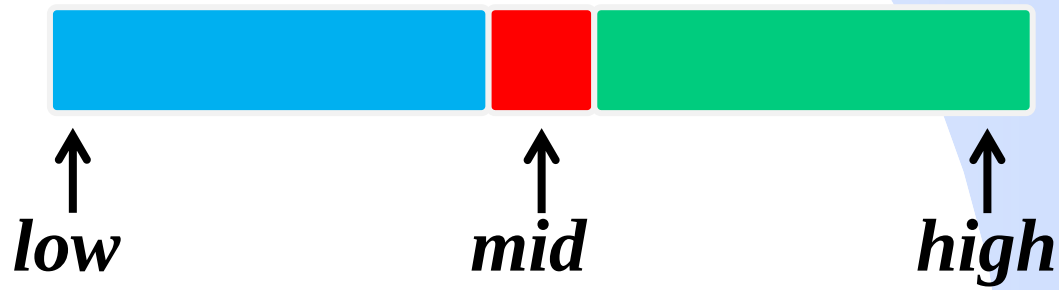


对半查找

A

```
int BinarySearch(int R[],int n, int K){
    //在数组R中对半查找K，R中关键词递增有序
    int low = 1, high = n, mid;
    while(low <= high){    //在R_low...R_high之间查找K
        mid=(low+high)/2;
        if(K<R[mid]) high=mid-1;          //在左半部分查找
        else if(K>R[mid]) low=mid+1;      //在右半部分查找
        else return mid;                  //查找成功
    }
    return -1;          //查找失败
}
```

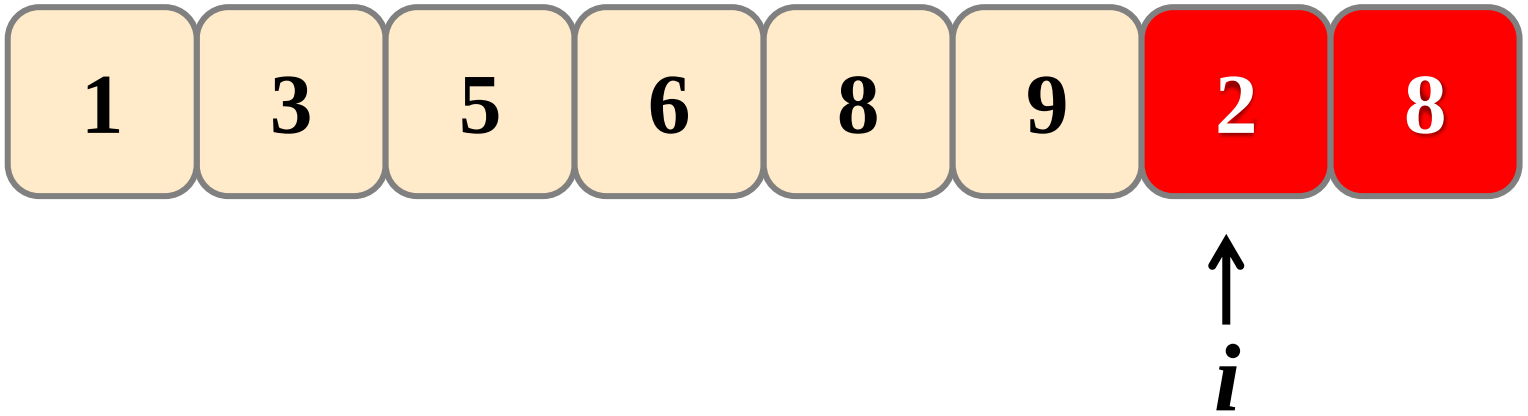
时间复杂度
O(log*n*)



对半插入排序

A

- 插入R_i时，基于对半查找确定插入的位置，可将每次插入的关键词比较次数降为O(log$n$)。
- 由于元素移动次数仍为O(n)，故排序算法总时间复杂度仍为O(n^2)，但常数更低。



练习

A

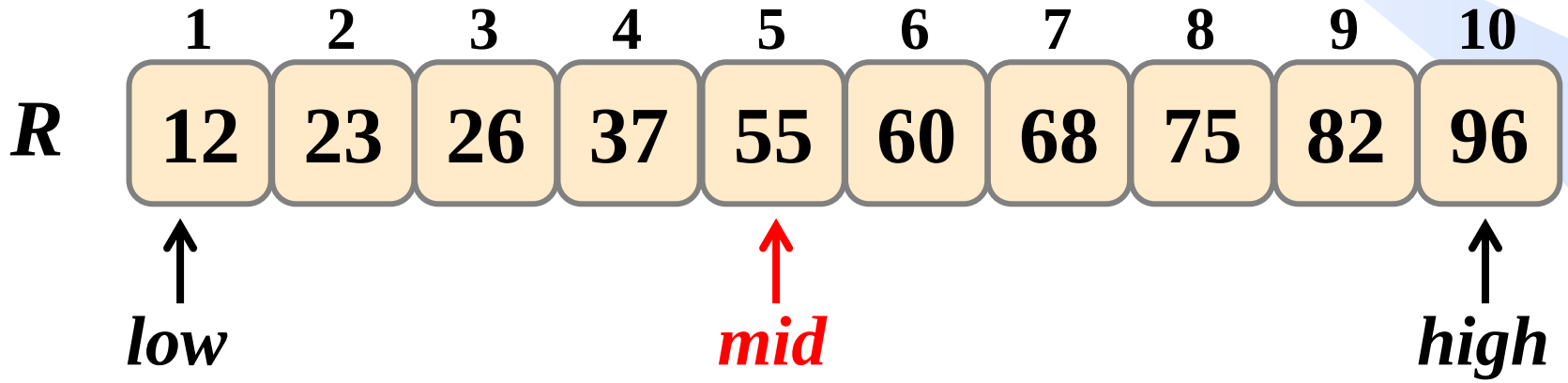
对同一序列分别进行对半插入排序和直接插入排序，两者之间可能的不同之处是_____。【考研题全国卷】

A.排序的总趟数　　B.元素的移动次数

C.使用辅助空间的数量　　D.元素的比较次数

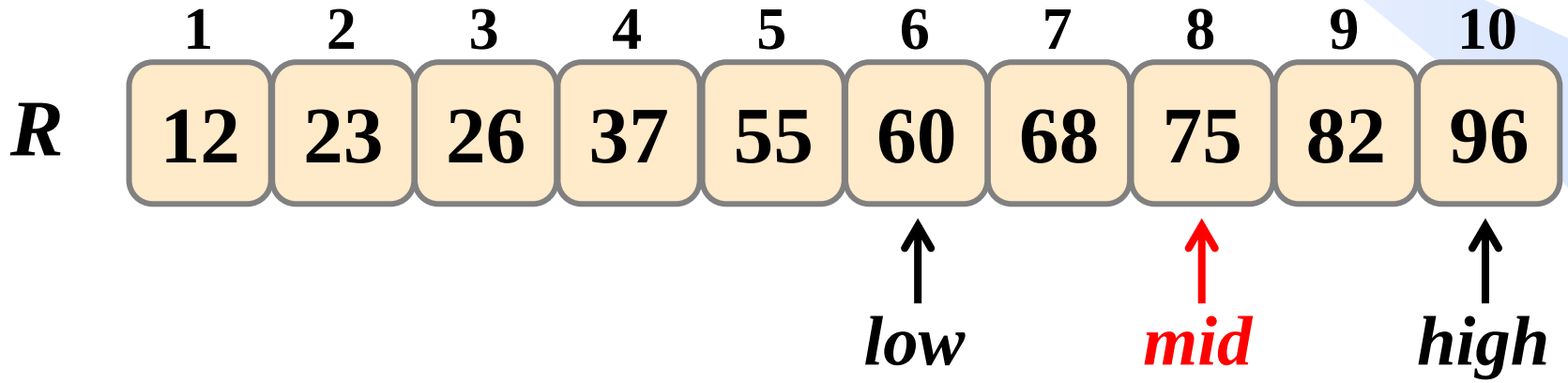
例：查找 K=96 时对半查找过程（第1次比较）

A

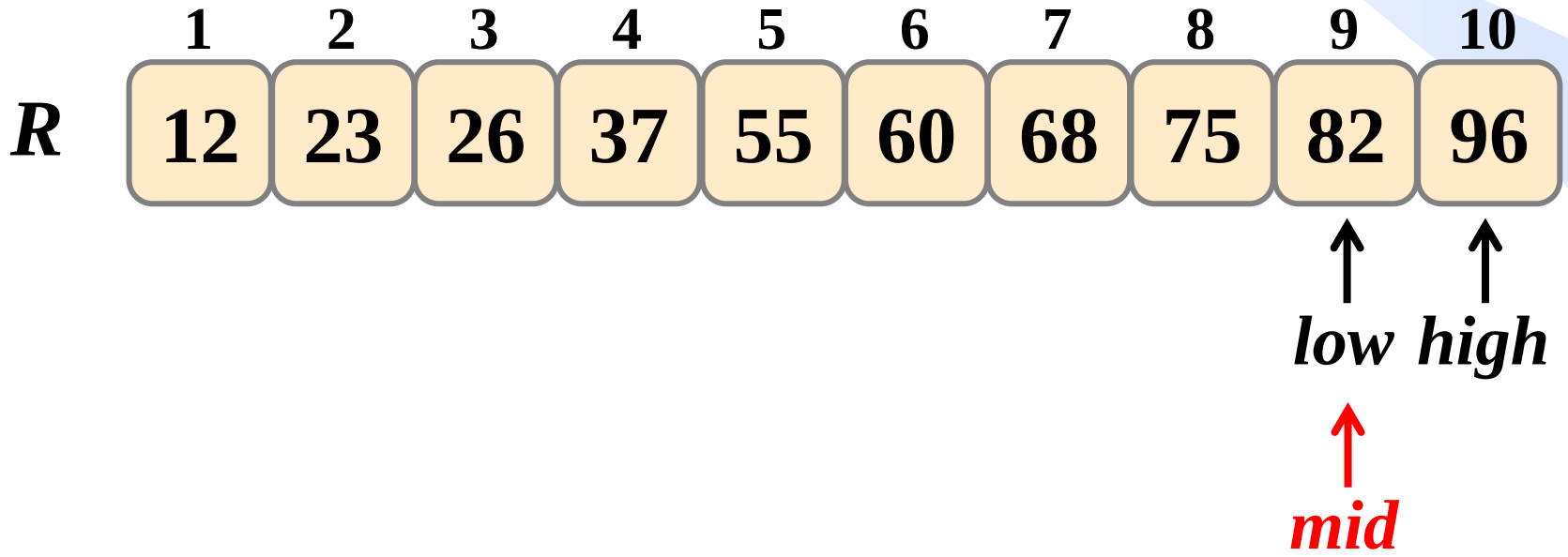


例：查找 K=96 时对半查找过程（第2次比较）

A

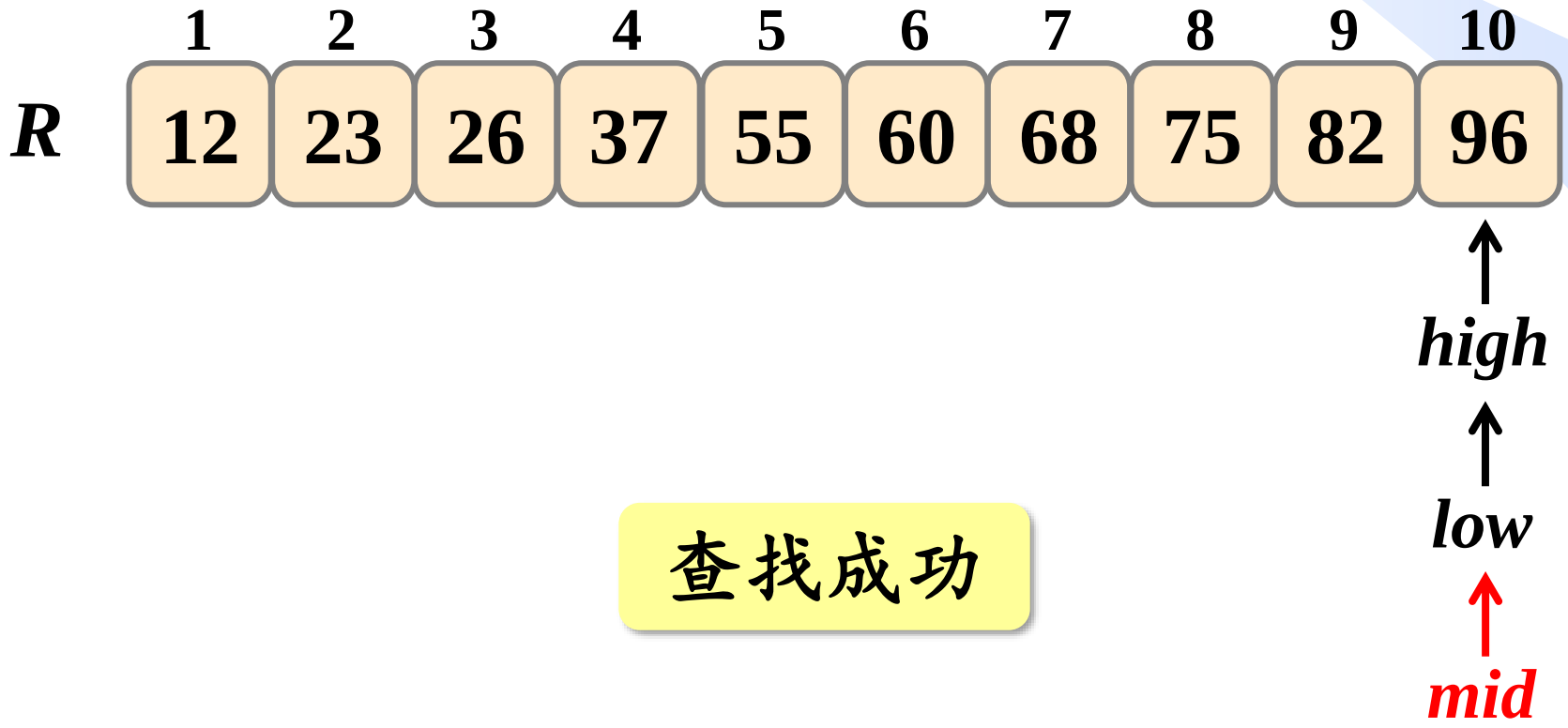


例：查找 *K*=96 时对半查找过程（第3次比较）



例：查找 K=96 时对半查找过程（第4次比较）

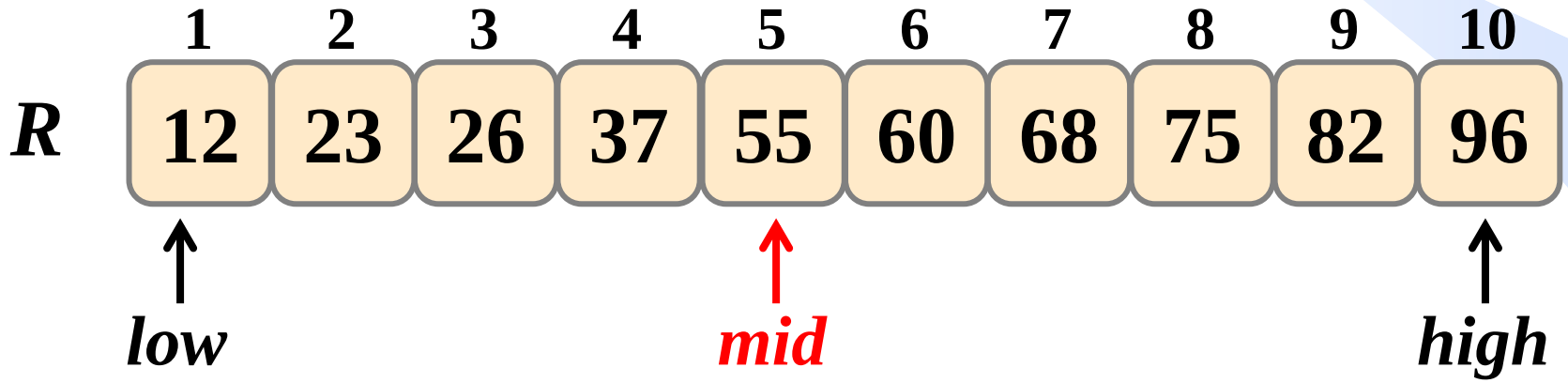
A



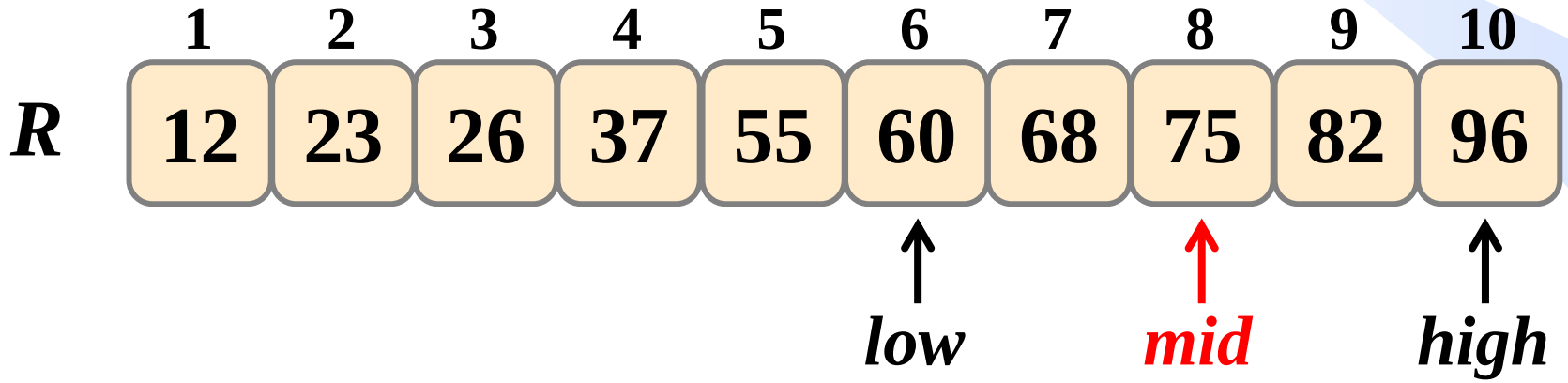
查找成功

14

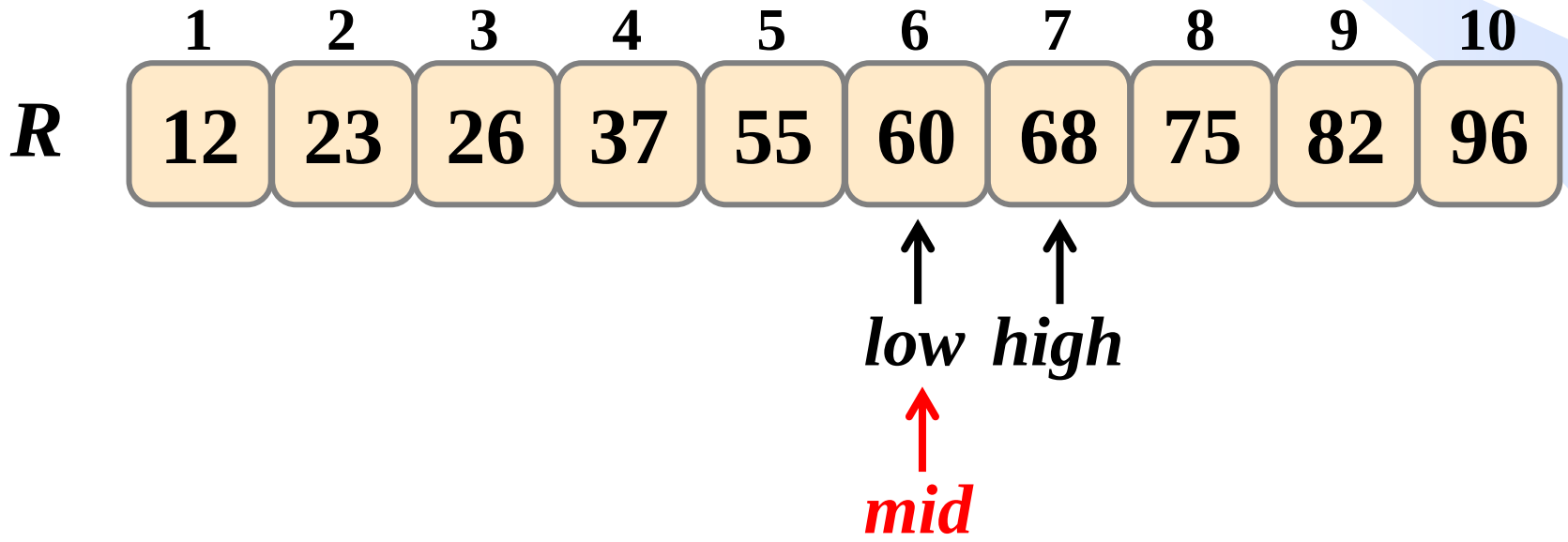
例：查找 K=58 时对半查找过程（第1次比较）



例：查找 K=58 时对半查找过程（第2次比较）

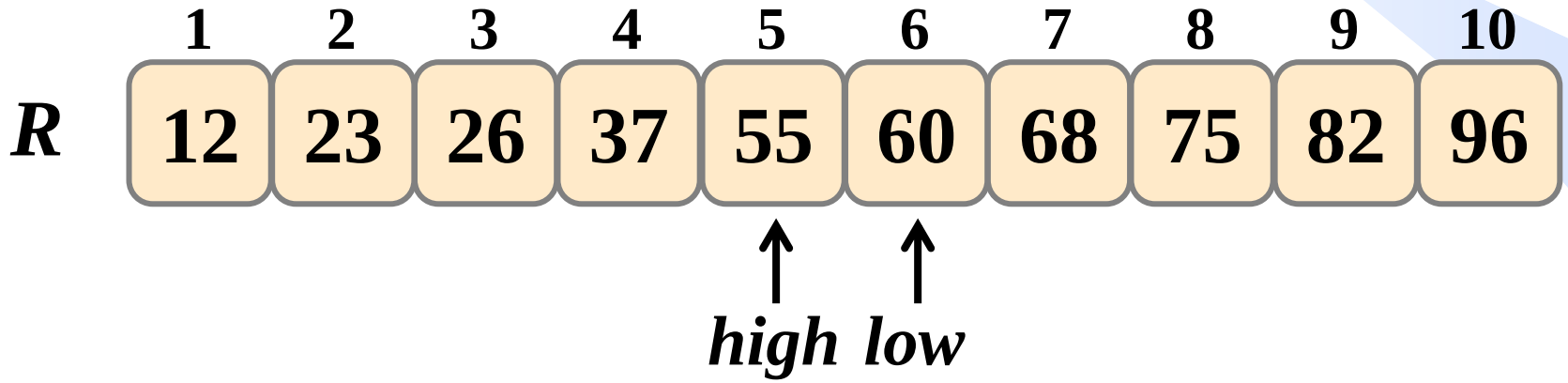


例：查找 K=58 时对半查找过程（第3次比较）



例：查找 K=58 时对半查找过程（第3次比较）

A



查找失败

18

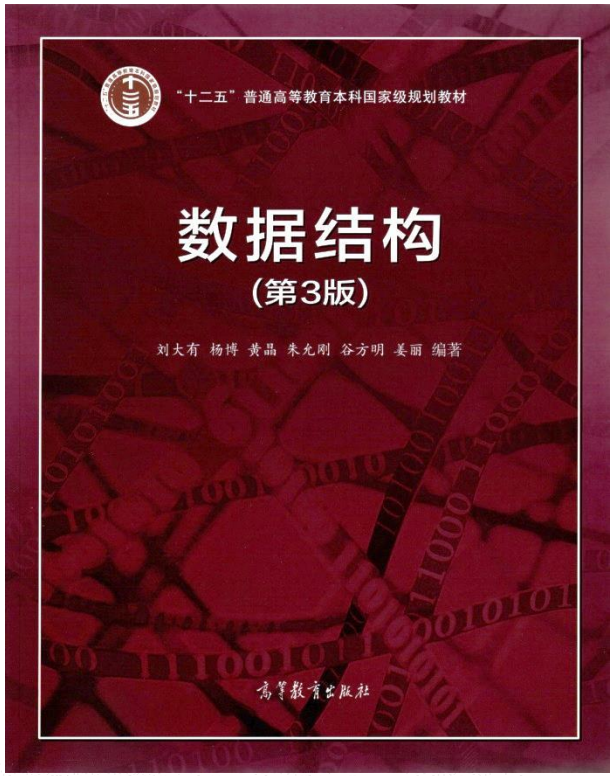
以下哪个分支语句执行效率更高？

D

```
switch(a){
    case 1000: f1(); break;
    case 2000: f2(); break;
    case 2500: f3(); break;
    case 5000: f5(); break;
    case 7000: f6(); break;
    case 7500: f7(); break;
    case 8000: f8(); break;
    case 9000: f9(); break;
    default: f10();
}
```

```
if(a==1000) f1();
else if(a==2000) f2();
else if(a==2500) f3();
else if(a==5000) f5();
else if(a==7000) f6();
else if(a==7500) f7();
else if(a==8000) f8();
else if(a==9000) f9();
else f10();
```

Visual Studio®



线性结构查找

顺序查找

对半查找

插值查找

再谈对半查找

数据之法

结构之美

算法之道

zhuyungang@jlu.edu.cn

Last updated on 2025.6

插值查找

A

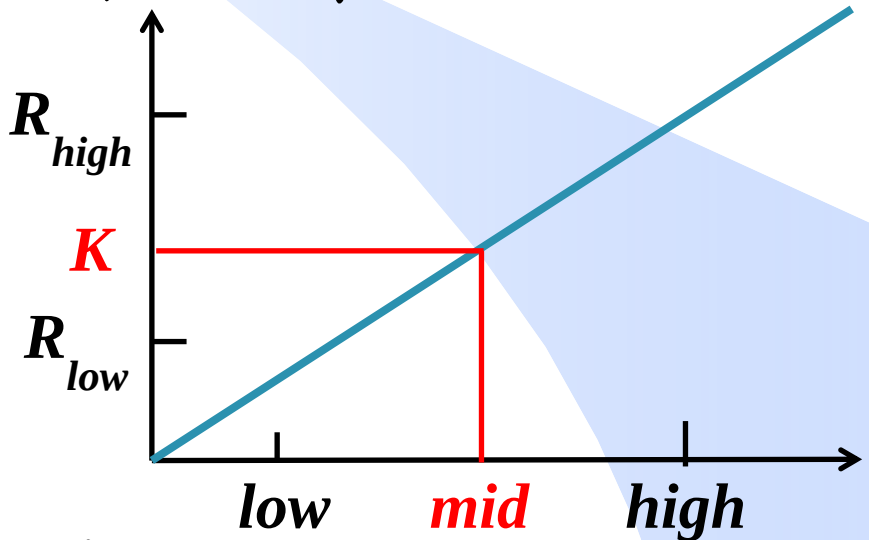
- 假设：有序数组R中元素均匀随机分布，例如

1	2	3	4	5	6	7	8	9	10
10	20	30	40	50	60	70	80	90	100

- 于是，$R_{low}...R_{high}$内各元素应大致呈线性增长关系

$$\frac{mid-low}{high-low} \approx \frac{K-R_{low}}{R_{high}-R_{low}}$$

$$mid \approx low + \frac{K-R_{low}}{R_{high}-R_{low}}(high-low)$$

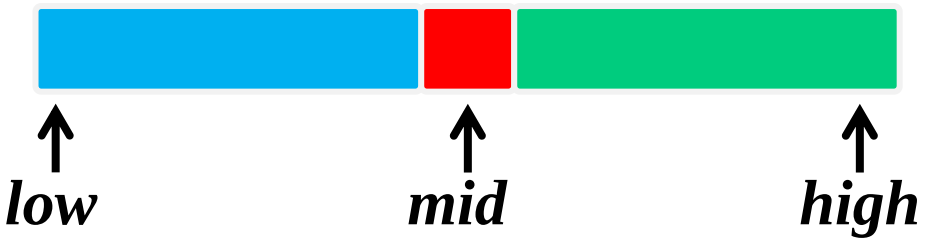


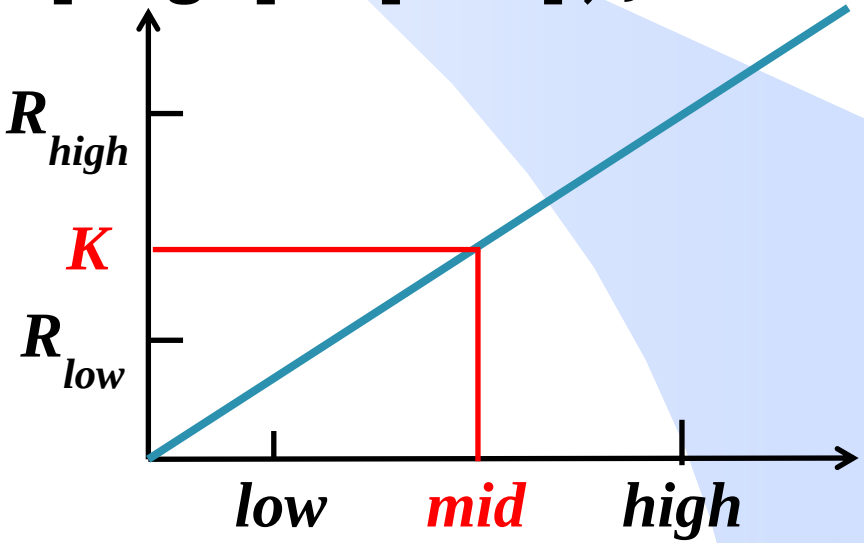
- 每次迭代过程中，通过线性插值预测K的期望位置mid。
- 回顾对半查找：$mid = \frac{low+high}{2} = low + \frac{1}{2}(high-low)$

插值查找

A

```
int InterpolationSearch(int R[], int n, int K){
    int low=1, high=n, mid;
    while(low<=high && K>=R[low] && K<=R[high]){
        if(R[low]==R[high]) return low;
        mid=low+(K-R[low])*(high-low)/(R[high]-R[low]);
        if(K<R[mid]) high = mid-1;
        else if(K>R[mid]) low = mid+1;
        else return mid;
    }
    return -1;
}
```





$$mid \approx low + \frac{K - R_{low}}{R_{high} - R_{low}}(high - low)$$


在英文词典查找单词“zoo”，你为什么不用对半查找法，而直接从字典的后面找？

插值查找时间复杂度

A

姚期智
哈佛大学博士
图灵奖获得者
中国科学院院士
美国科学院外籍院士
加州伯克利教授（1981-1982）
斯坦福大学教授（1982-1986）
普林斯顿大学教授（1986-2004）
清华大学教授（2004-现在）

储枫
麻省理工学院博士
清华大学教授

姚期智及夫人储枫证明：插值查找算法每经一次迭代，平均情况下待查找区间的长度由n缩至$\sqrt{n}$。

AC Yao and FF Yao. The Complexity of Searching an Ordered Random Table. *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*. 173-177, 1976.

插值查找时间复杂度

A

初始长度	第1次迭代后	第2次迭代后	第3次迭代后	……	第k次迭代后	第k+1次迭代后
n	$\sqrt{n}$	$\sqrt{\sqrt{n}}$	$\sqrt{\sqrt{\sqrt{n}}}$	……	2	1

- 元素比较的次数 = 迭代的次数。
- $n^{(\frac{1}{2})^k} = 2$
- 平均时间复杂度O(loglog$n$).
- 最坏情况：元素分布极不均匀；
  最坏时间复杂度O(n).

1	2	3	5	6	9999
1	2	3	4	5	6

插值查找总结

- 从O(log*n*)到O(loglog*n*)优势并不明显（除非查找表极长，或比较操作成本极高）。

  比如$n=2^{32}\approx 42.9$亿

  $\log n = \log 2^{32} = 32$

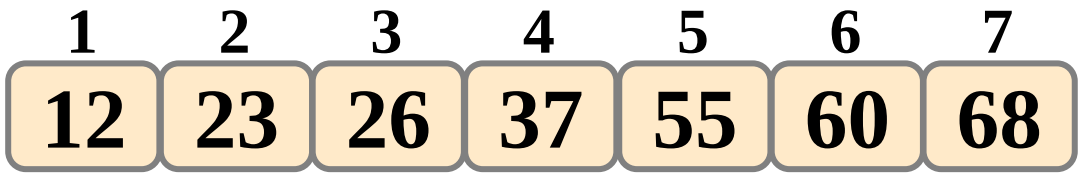
  $\log\log n = \log 32 = 5$

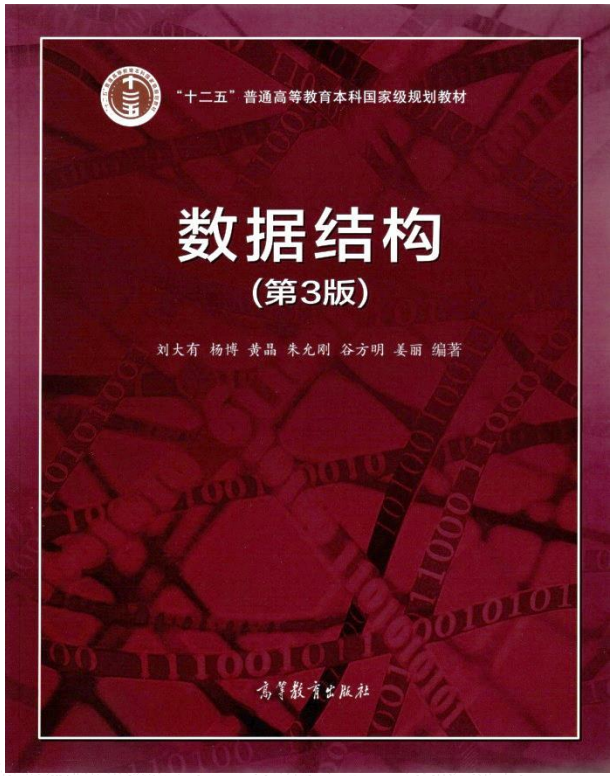
- 需引入乘除法运算。
- 元素分布不均匀时效率受影响。
- 实际中可行的方法：首先通过插值查找迅速将查找范围缩小到一定的范围，然后再进行对半查找或顺序查找。

二分查找总结

A

- 优点：平均查找效率不超过O(log*n*)，比顺序查找高。
- 缺点：
  - ✓适用于有序数组，对有序链表难以进行二分查找。
  - ✓适用于静态查找场景，若元素动态变化（频繁增删）后，为了维持数组有序，需要O(*n*)时间调整，与顺序查找相比，就没有优势了。





线性结构查找

顺序查找

对半查找

插值查找

再谈对半查找

数据之法

结构之美

算法之道

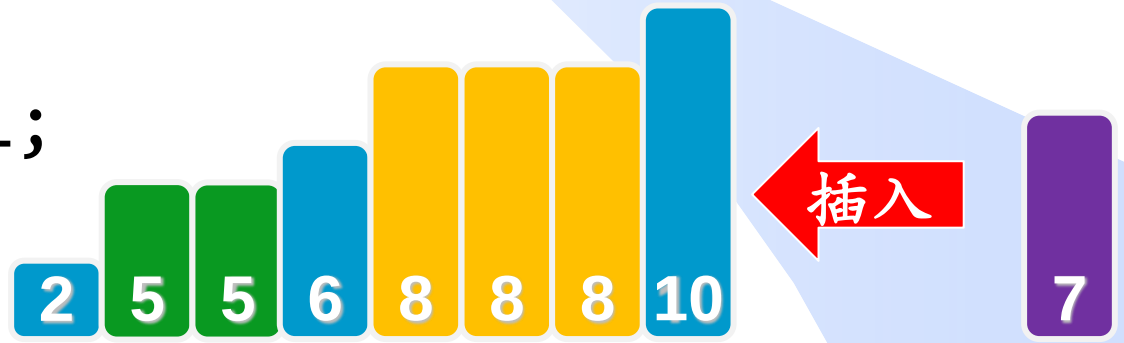
zhuyungang@jlu.edu.cn

Last updated on 2025.6

回顾——传统对半查找

C

```
int BinarySearch(int R[],int n, int K){
    //在数组R中对半查找K，R中关键词递增有序
    int low = 1, high = n, mid;
    while(low <= high){
        mid=(low+high)/2;
        if(K<R[mid]) high=mid-1;
        else if(K>R[mid]) low=mid+1;
        else return mid;
    }
    return -1; //查找失败
}
```



更好的方案：返回更有价值信息

- 若查找失败，能给出查找失败的位置，便于新元素插入

返回：

（1）小于等于K的最后一个位置

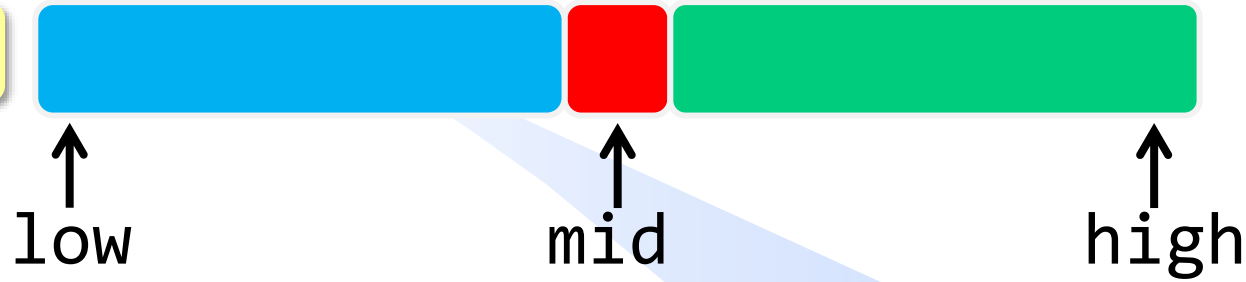
（2）大于等于K的第一个位置

进一步审视对半查找过程

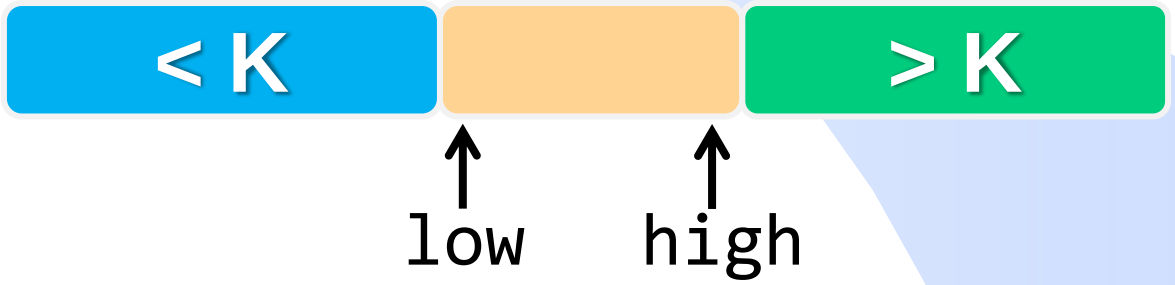
C

```
int BinarySearch(int R[],int n, int K){
    int low=1, high=n, mid;
    while(low <= high){
        mid=(low+high)/2;
        if(K < R[mid])
            high = mid-1;
        else if(K > R[mid])
            low = mid+1;
        else
            return mid;
    }
    return -1; //查找失败
}
```

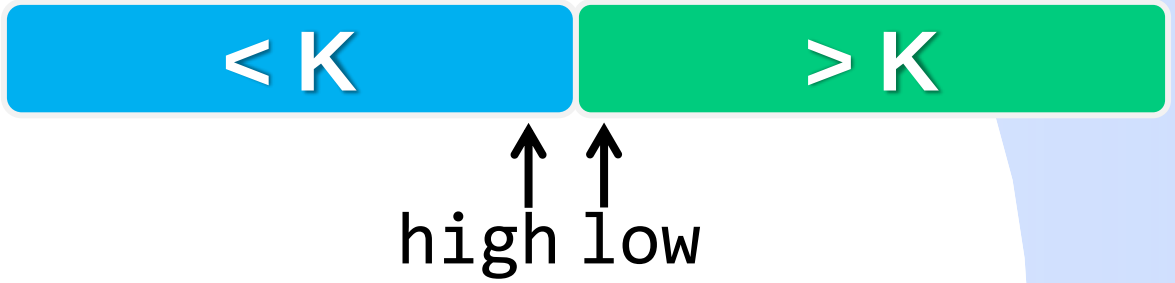
初始时



执行过程中



查失败结束后

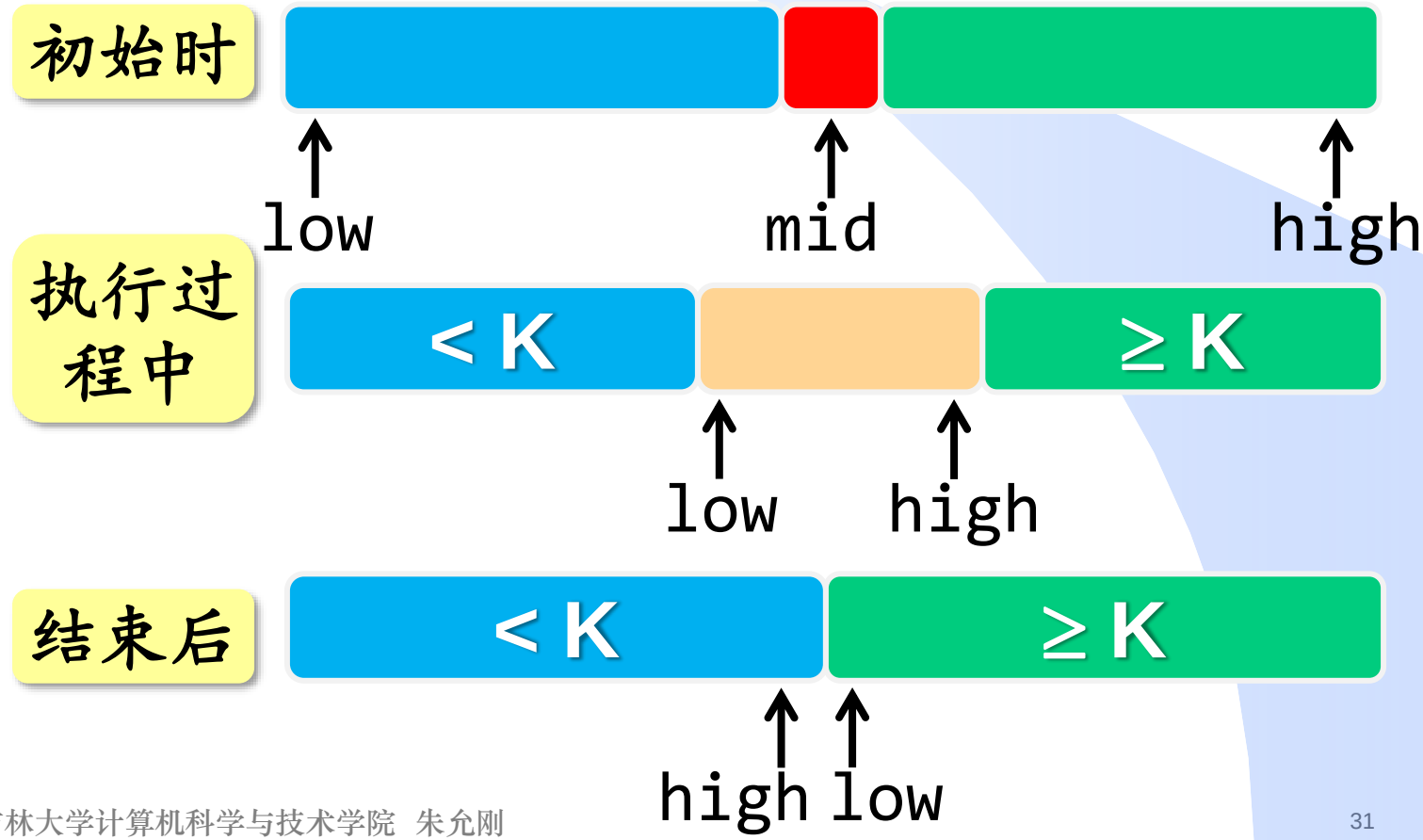


low的左边<K，high的右边>K

大于等于K的第一个位置

C

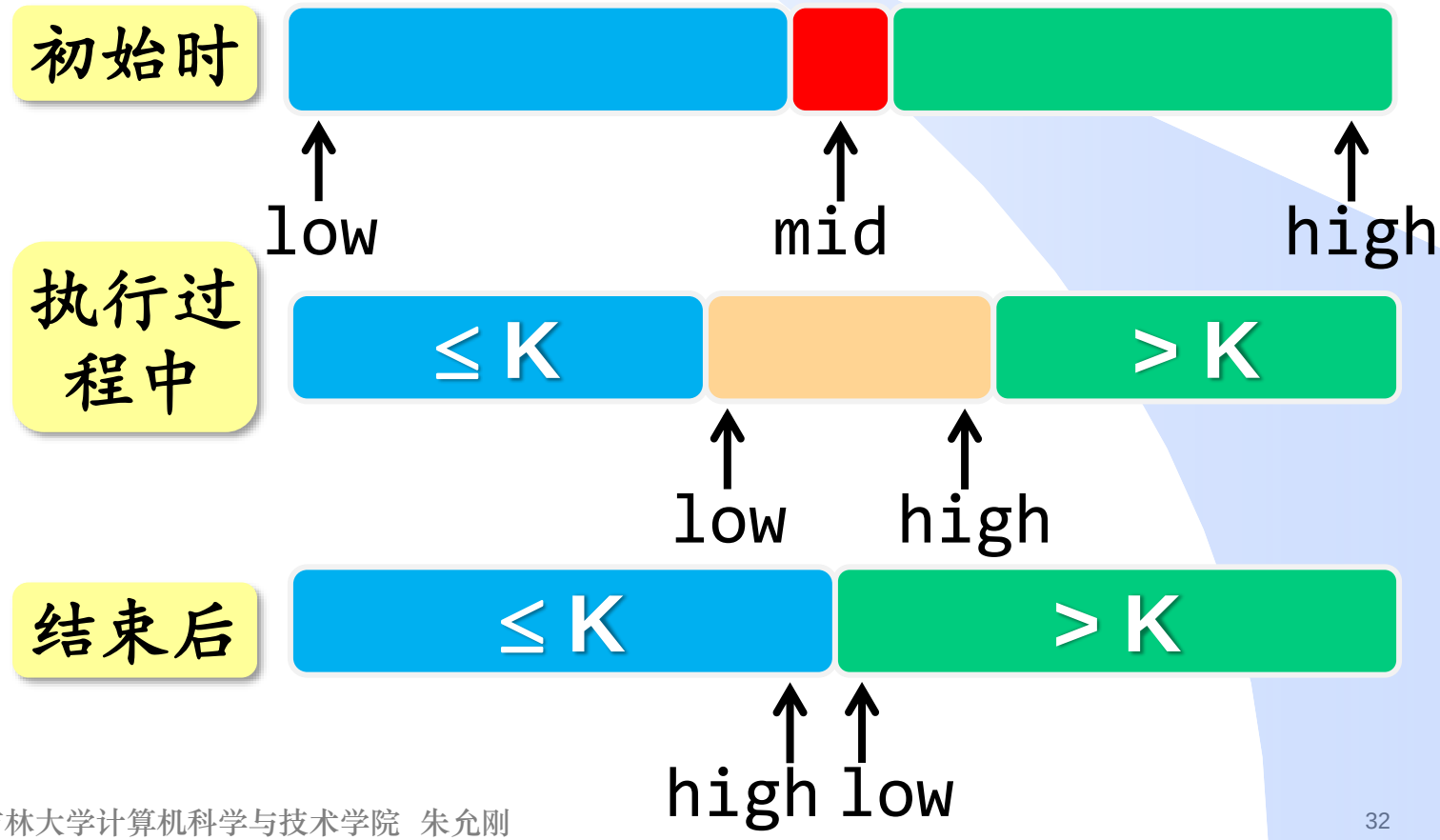
```
int Lower_bound(int R[], int low, int high, int K){
    while(low <= high){
      int mid=(low+high)/2;
      if(K <= R[mid])
          high = mid-1;
      else
          low = mid+1;
    }
    return low;
}
```



小于等于K的最后一个位置

C

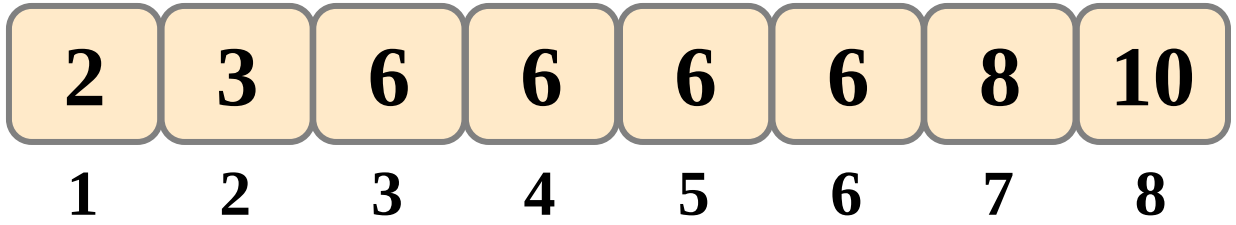
```
int Upper_bound(int R[], int low, int high, int K){
    while(low <= high){
      int mid=(low+high)/2;
      if(K < R[mid])
          high = mid-1;
      else
          low = mid+1;
    }
    return high;
}
```



例题：第一个/最后一个等于K的位置

C

给定有序整型数组R和一个整数K，找出K在数组中开始位置和结束位置，若K不在R中则返回-1，数组下标从0开始。【华为、字节跳动、百度、阿里、美团、小米、360、谷歌、微软、苹果面试题LeetCode34】



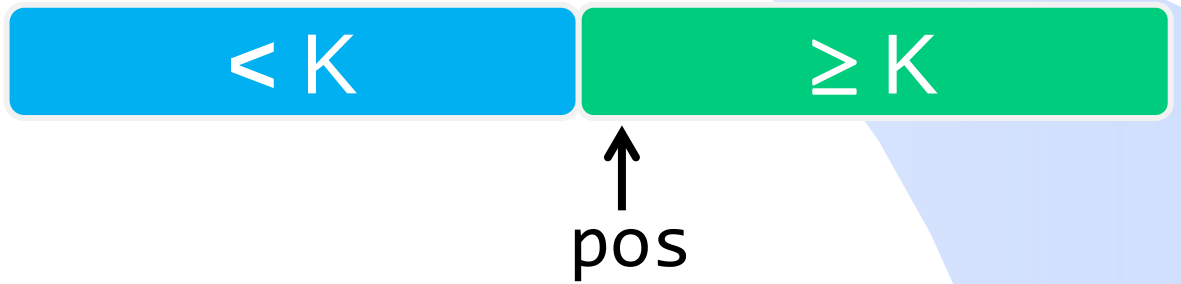
第一个等于K的位置

C

策略：先找≥K的第一个位置，再看该位置的元素是否等于K

```
int LeftBound(int R[], int n, int K){
      int pos = Lower_bound(R,0,n-1,K);
      if(pos>=0 && pos<n && R[pos]==K) return pos;
      return -1;
}
```

时间复杂度
O(log*n*)




```
int Lower_bound(int R[], int low, int high, int K)
```

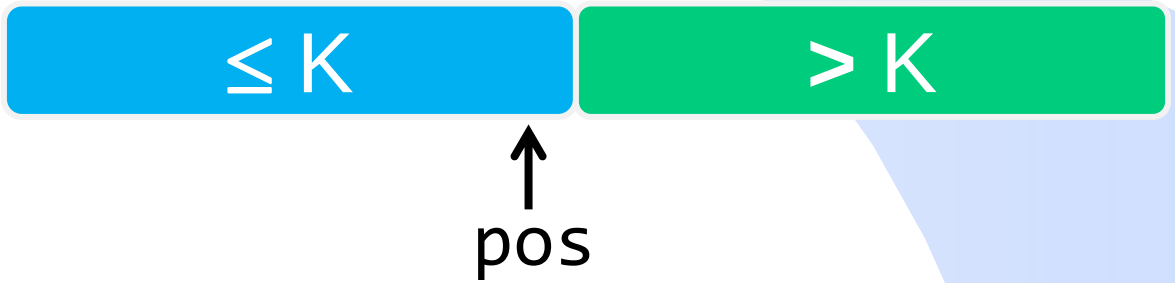
最后一个等于K的位置

C

策略：先找≤K的最后一个位置，再看该位置元素是否等于K

```
int RightBound(int R[], int n, int K){
      int pos = Upper_bound(R,0,n-1,K);
      if(pos>=0 && pos<n && R[pos]==K) return pos;
      return -1;
}
```

时间复杂度
O(log*n*)




```
int Upper_bound(int R[], int low, int high, int K)
```