快速选择和二分法

- **快速选择算法**
- 二分查找拓展

数据之法
结构之美
算法之道

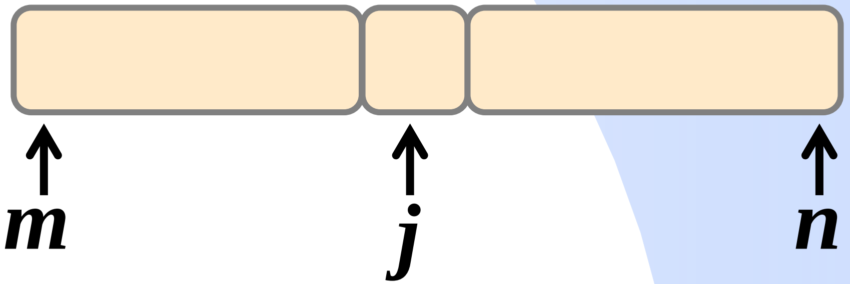
Last updated on 2024.12

zhuyungang@jlu.edu.cn

快速选择问题

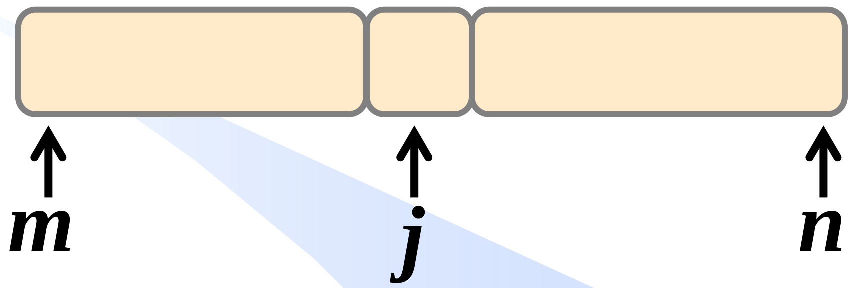
给定包含n个元素的整型数组，编写算法找数组中第$k(k<n)$小的数，要求时间复杂度为$O(n)$。【腾讯、字节跳动、阿里、美团、京东、滴滴、网易、小米、华为、旷视科技、拼多多、苹果、微软、百度、谷歌、快手面试题】

- 基于快速排序的Partition算法，将数组分成两部分。
- 若左区间元素个数等于k-1，则基准元素即为第k小的元素。
- 若左区间元素个数大于k-1，第k小的元素必在左区间，对左区间递归找第k小的元素。
- 若左区间长度小于k-1，第k小的元素必在右区间，若左区间元素个数为n_L，则在右区间递归找第k-n_L-1小的元素。



快速选择问题

```
int QuickSelect(int R[],int m, int n, int k){
    int j=Partition(R, m, n);
    int nL=j-m;      //左部分元素个数
    if(nL==k-1) return R[j];
    else if(nL>k-1) //在左部分找第k小数
        return QuickSelect(R,m,j-1,k);
    else             //在右部分找第k-nL-1小数
        return QuickSelect(R, j+1, n, k-nL-1);
}
```

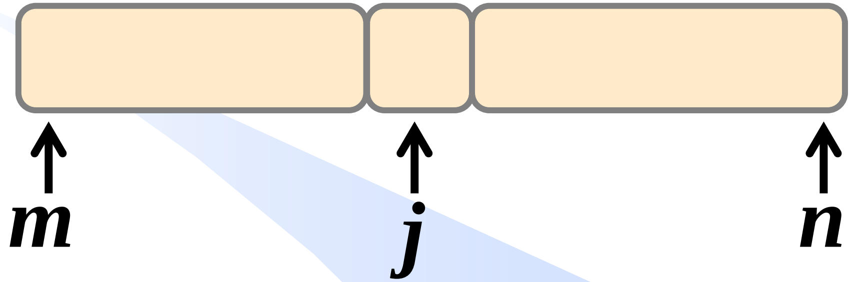


与快速排序不同在于，只需递归处理左右两个子数组中的一个

减而治之，减治法

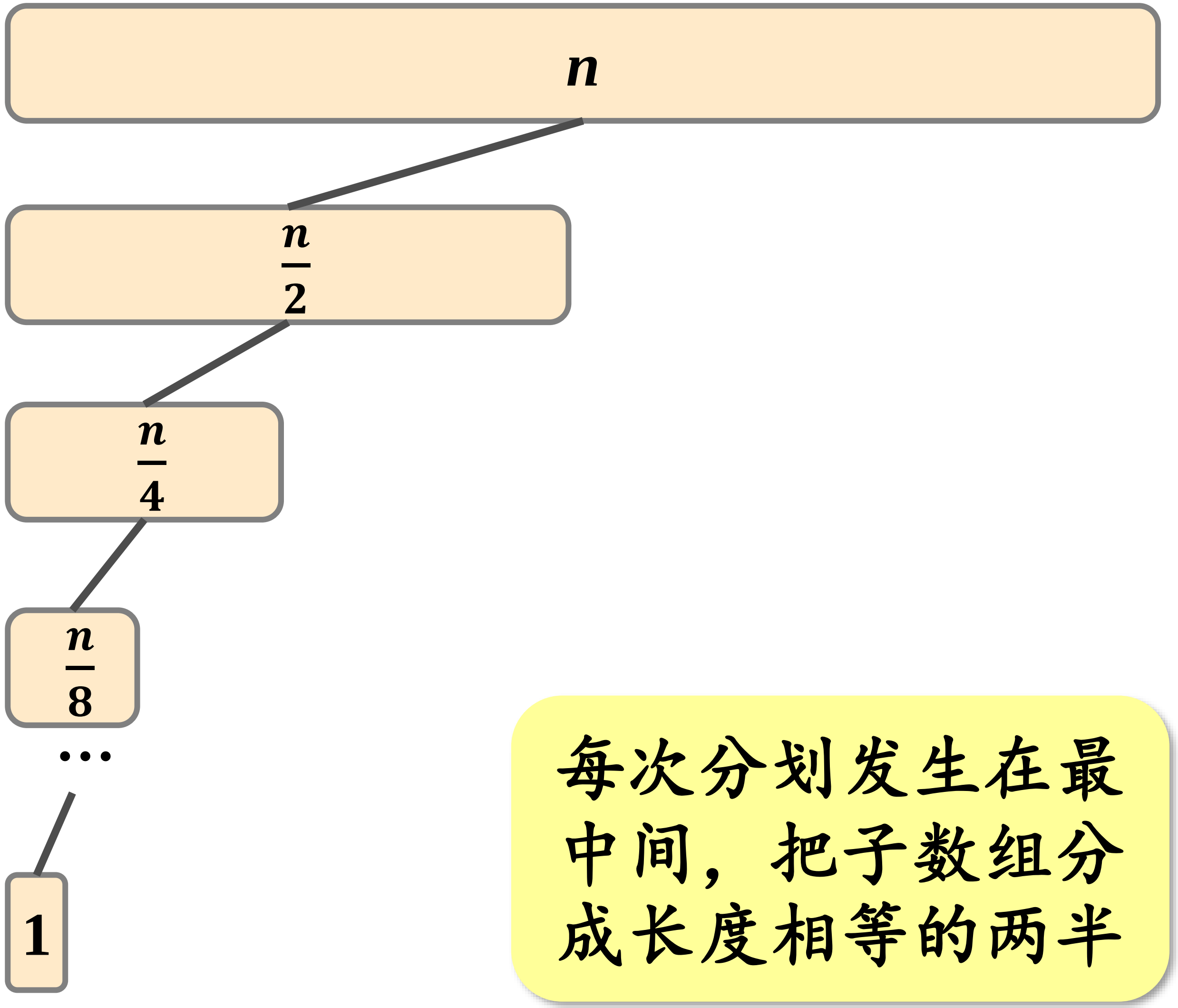
快速选择问题——迭代形式

```
int QuickSelect(int R[],int m, int n, int k){
    while(true){
        int j=Partition(R, m, n);
        int nL=j-m;        //左部分元素个数
        if(nL==k-1) return R[j];
        else if(nL>k-1) n=j-1; //在左部找第k小数
        else m=j+1,k=k-nL-1;   //在右部找第k-nL-1小数
    }
}
```



初始调用QuickSelect(R, 1, n, k).

最好/平均情况时间复杂度

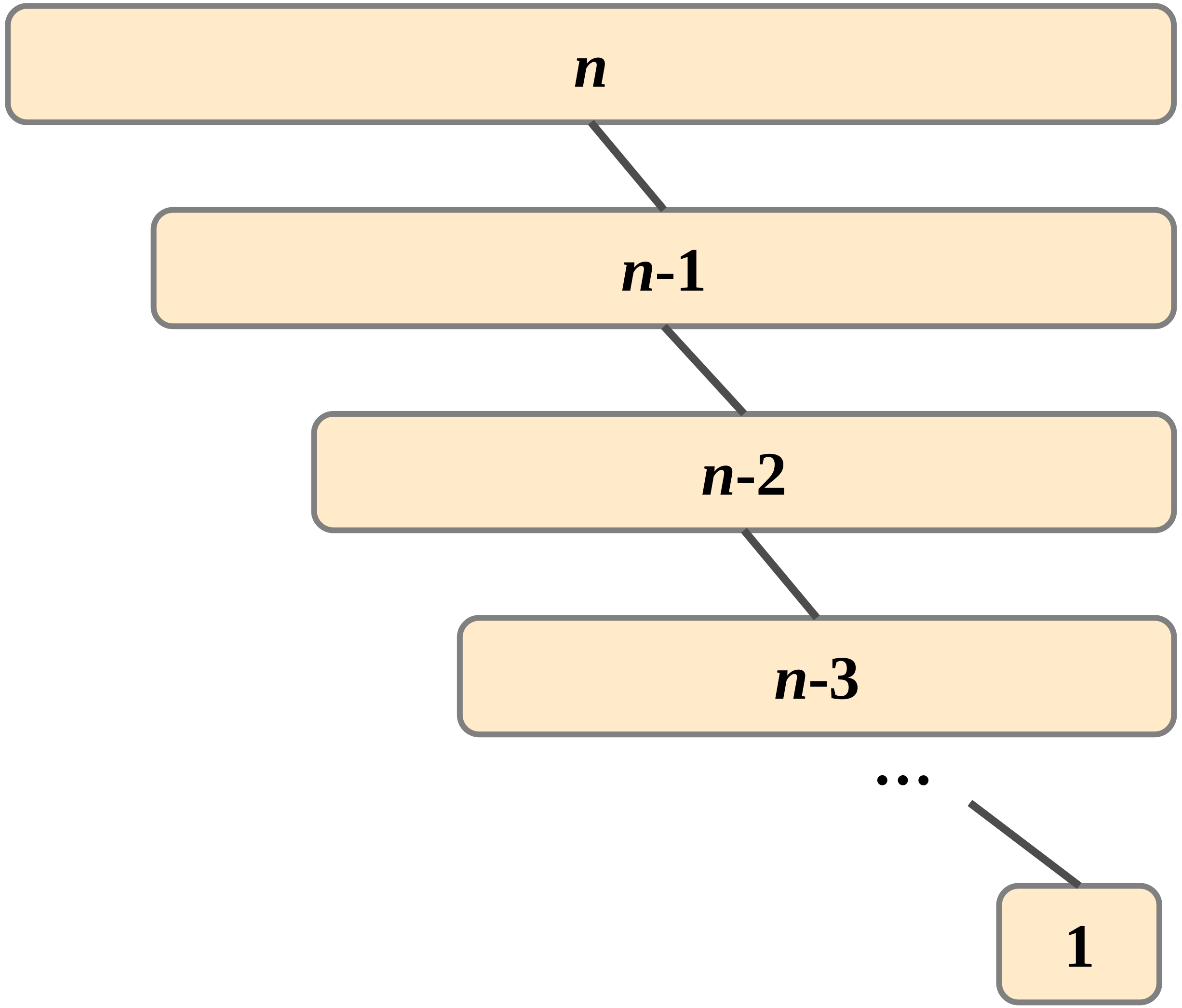


$$
\begin{aligned}
\mathrm{T}(n) &= n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 1 \\
&= n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots\right) \\
&= 2n\left(1 - \frac{1}{2^k}\right) \\
&\le 2n \\
&= \mathrm{O}(n)
\end{aligned}
$$

每次分划发生在最中间，把子数组分成长度相等的两半

时间复杂度 O(n)

最坏情况时间复杂度



分划极不平衡，每次分划都发生在子数组最边上，如初始时已排好序或逆序

时间复杂度
$O(n^2)$

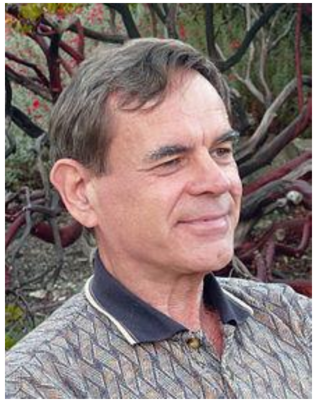
优化策略：随机选取基准元素。
降低最坏情况发生概率，无法杜绝

最坏时间为O(*n*)的选择算法

中位数的中位数（Median of Medians）算法，也称BFPRT算法。由如下5位学者提出：

Manuel Blum
图灵奖获得者
美国科学院院士
美国工程院院士
卡内基梅隆大学教授

Robert Floyd
图灵奖获得者
斯坦福大学教授



Vaughan Pratt
斯坦福大学教授

Ronald Rivest
图灵奖获得者
美国科学院院士
美国工程院院士
麻省理工学院教授

Robert Tarjan
图灵奖获得者
美国科学院院士
美国工程院院士
普林斯顿大学教授

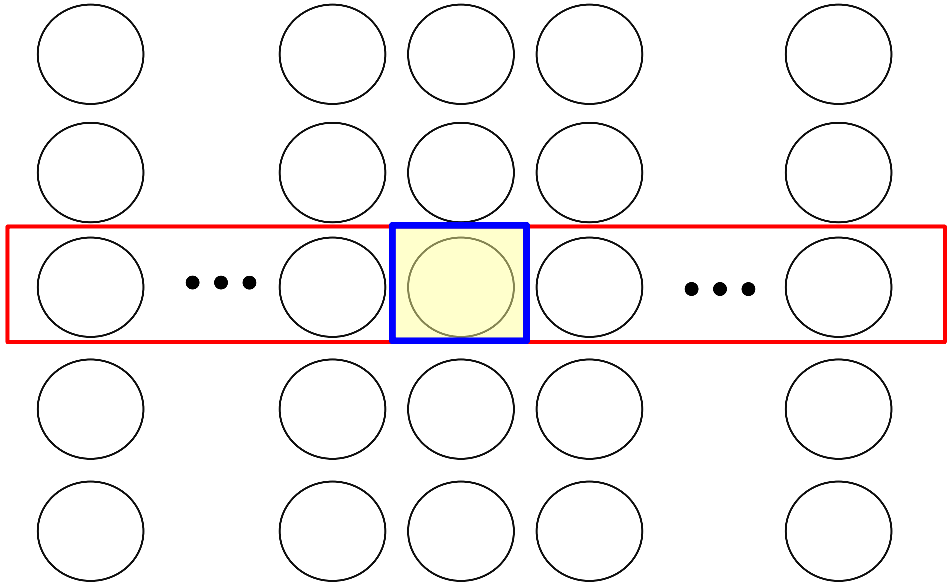
Median of Medians算法

算法SELECT

（1）将数组的n个元素划分为$\lceil n/5 \rceil$组，每组5个元素（最后一组可能不足5个）；

（2）找出每一组的中位数：对每组元素进行插入排序，排序后第3小的元素即中位数。

（3）对上步找出的$\lceil n/5 \rceil$个中位数，递归调用SELECT找出其中位数。



合理选择基准元素，使分划不至于太不均衡。

Median of Medians算法

算法SELECT

（1）将数组的n个元素划分为$\lceil n/5 \rceil$组，每组5个元素（最后一组可能不足5个）；

（2）找出每一组的中位数：对每组元素进行插入排序，排序后第3小的元素即中位数。

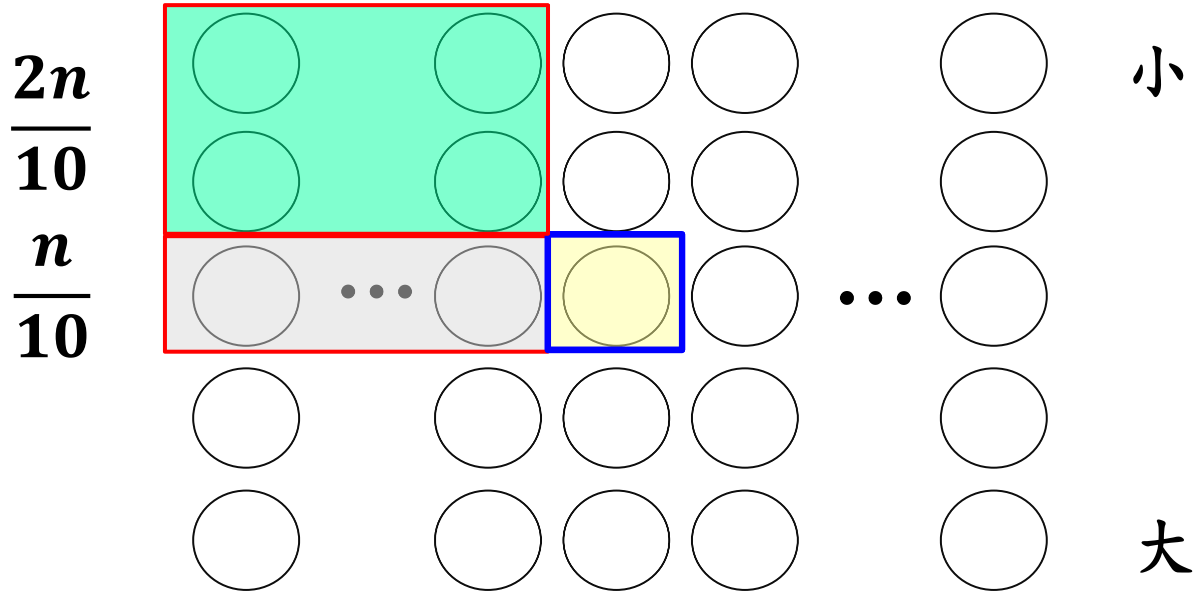
（3）对上步找出的$\lceil n/5 \rceil$个中位数，递归调用SELECT找出其中位数。

（4）将该元素作为基准元素，进行Partition，将数组划分为左右两部分，左部分元素个数为n_L。

（5）若$n_L=k-1$，算法结束；若n_L大于$k-1$，对左部分递归调用SELECT找第k小元素；若n_L小于$k-1$，对右部分递归调用SELECT找第$k-n_L-1$小元素。

最坏情况时间复杂度分析

从理论上保证分划不会过于不平衡，最差也能将数组分为3:7。



最坏情况时间复杂度分析

$T(n) \le T(0.2n)+ T(0.7n)+cn$

算法SELECT　　　　T(n)

（1）将数组的n个元素划分为$\lceil n/5 \rceil$组，每组5个元素（最后一组可能不足5个）；　　O(n)

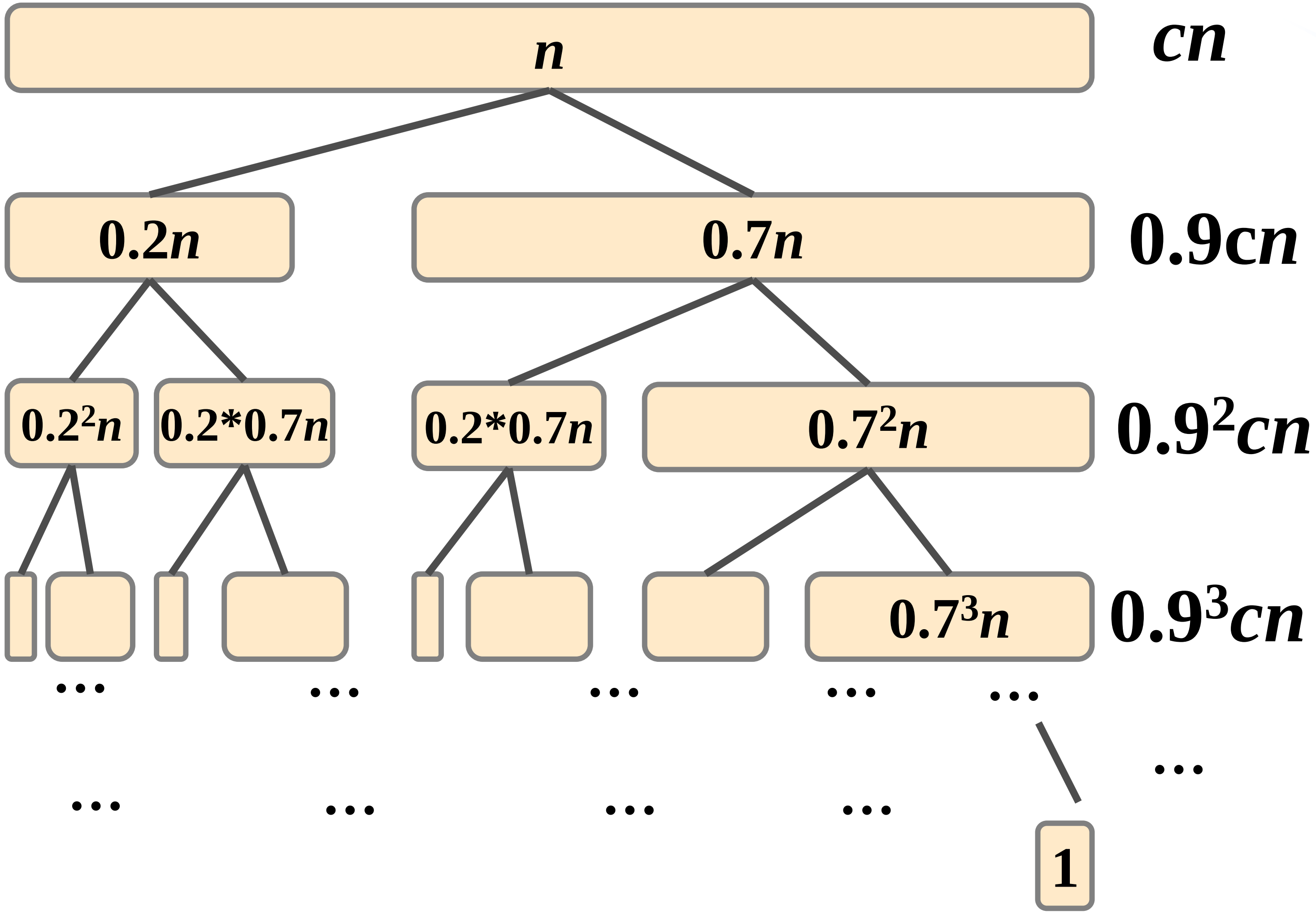
（2）找出每一组的中位数：对每组元素进行插入排序，排序后第3小的元素即中位数。　　O(n)

（3）对上步找出的$\lceil n/5 \rceil$个中位数，递归调用SELECT算法，找出其中位数。　　T(n/5)=T(0.2n)

（4）将该元素作为基准元素，进行Partition，将数组划分为左右两部分，左部分元素个数为n_L。　　O(n)

（5）若n_L=k-1，算法结束；若n_L大于k-1，对左部分递归调用SELECT找第k小元素；若n_L小于k-1，对右部分递归调用SELECT找第k-n_L-1小元素。　　T(7n/10) =T(0.7n)

最坏情况时间复杂度分析



$T(n) \le T(0.2n) + \mathrm{T}(0.7n) + cn$

$T(n) \le c(n + 0.9n + 0.9^2 n + 0.9^3 n + \ldots)$

$$= \frac{cn(1-0.9^k)}{1-0.9}$$

$$= 10cn(1-0.9^k)$$

$$\le 10cn$$

$$= O(n)$$

快速排序的另一优化策略

- 快速排序算法中，可以利用BFPRT算法在最坏O(n)时间内，选取数组的中位数作为基准元素。使快速排序最坏情况时间复杂度降为O($n\log n$)。
- 但在实际应用中，由于该算法较耗时，复杂度的常数较高，很多情况下速度不如堆排序或随机选取基准元素的快速排序，故实际很少使用该策略。

将数组中*k*个最小的元素全部输出

- 找到第*k*小的元素A后，再扫描一遍数组，输出数组中小于等于A的元素即可，时间代价O(*n*)。
- 数组中*k*个最大的元素，同理。

快速选择和二分法

- 快速选择算法
- 二分查找拓展

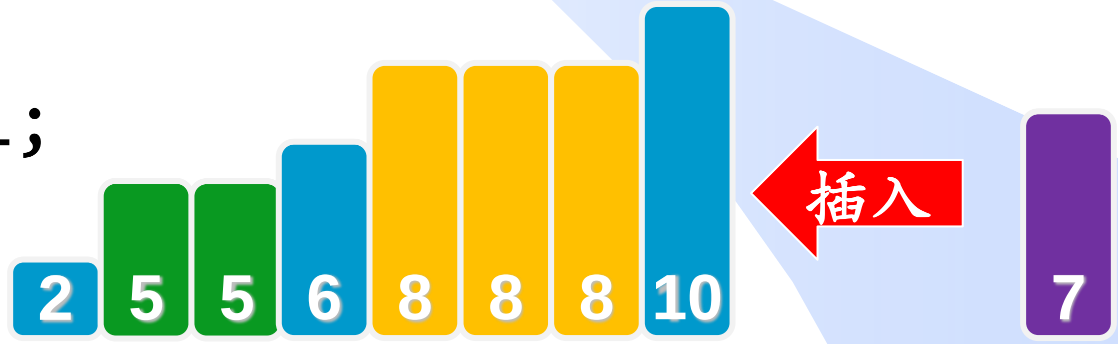
数据之法
结构之美
算法之道

Last updated on 2024.12

zhuyungang@jlu.edu.cn

回顾——传统对半查找

```
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：

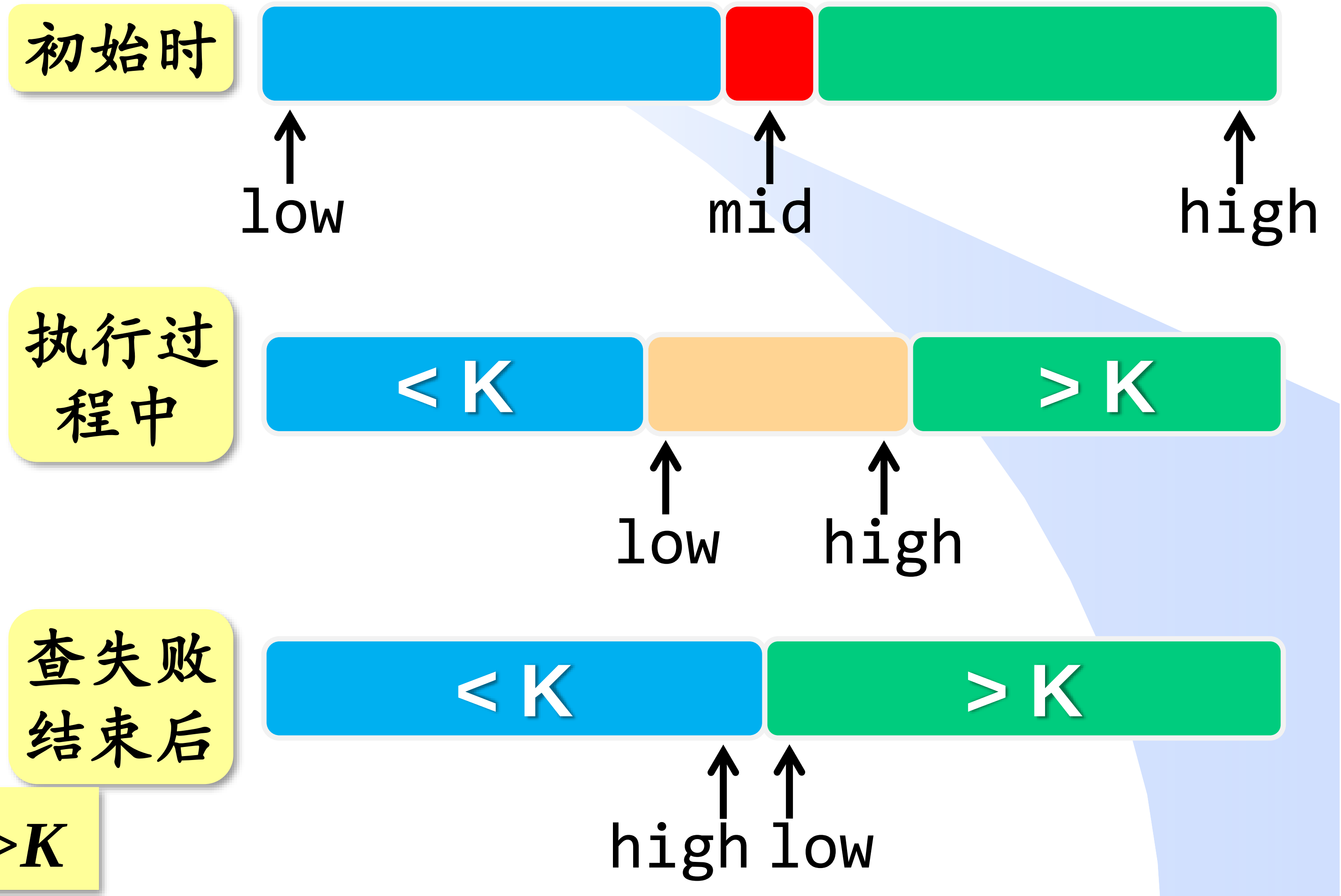
（1）小于等于K的最后一个位置

（2）大于等于K的第一个位置

进一步审视对半查找过程

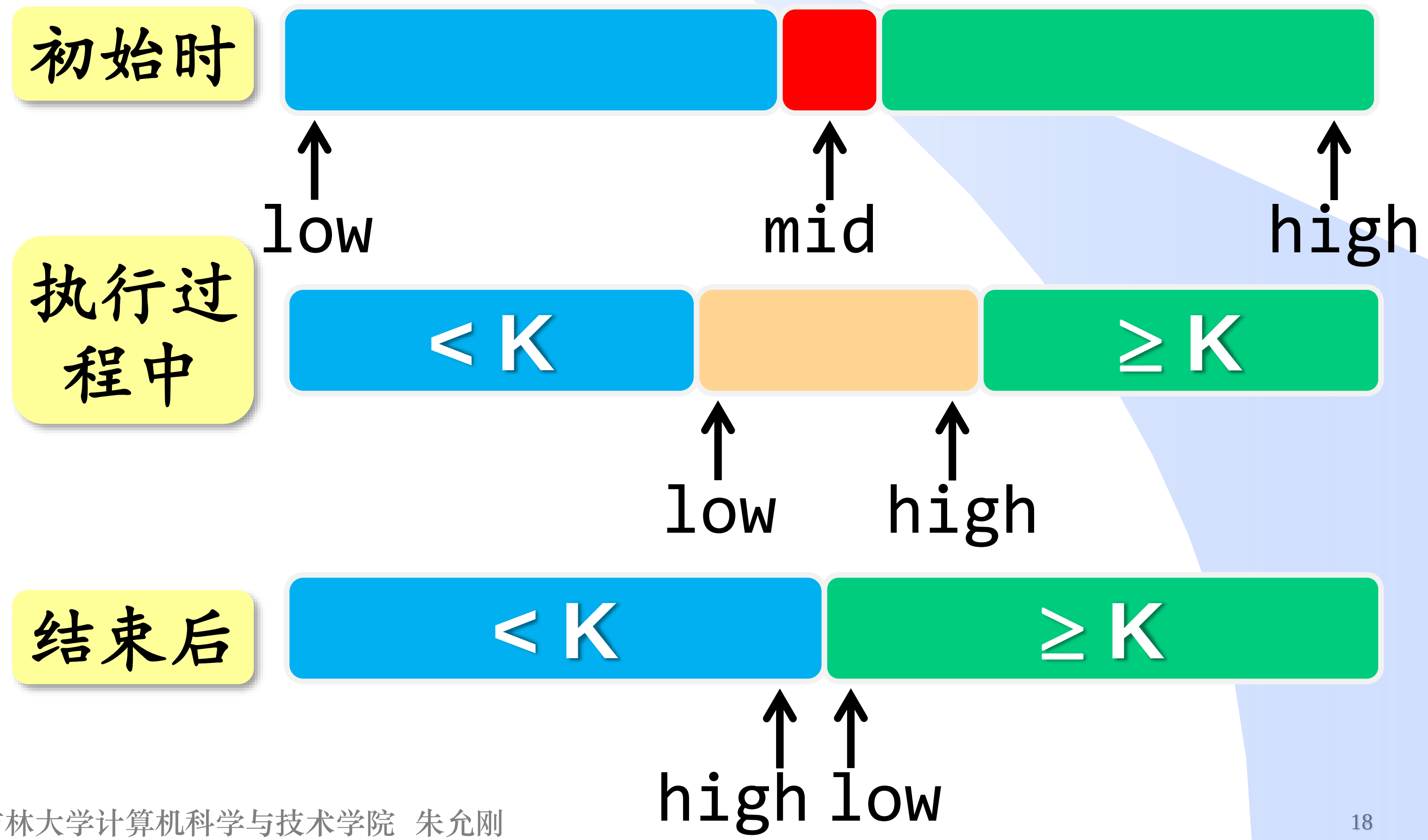
```
int BinarySearch(int R[],int n, int K){
    int low=1, high=n, mid;
    while(low <= high){
       mid=(low+high)/2;
       if(K < R[mid])
          high = mid-1;
       else if(K > R[mid])
          low = mid+1;
       else
          return mid;
    }
    return -1; //查找失败
}
```

low的左边<K，high的右边>K



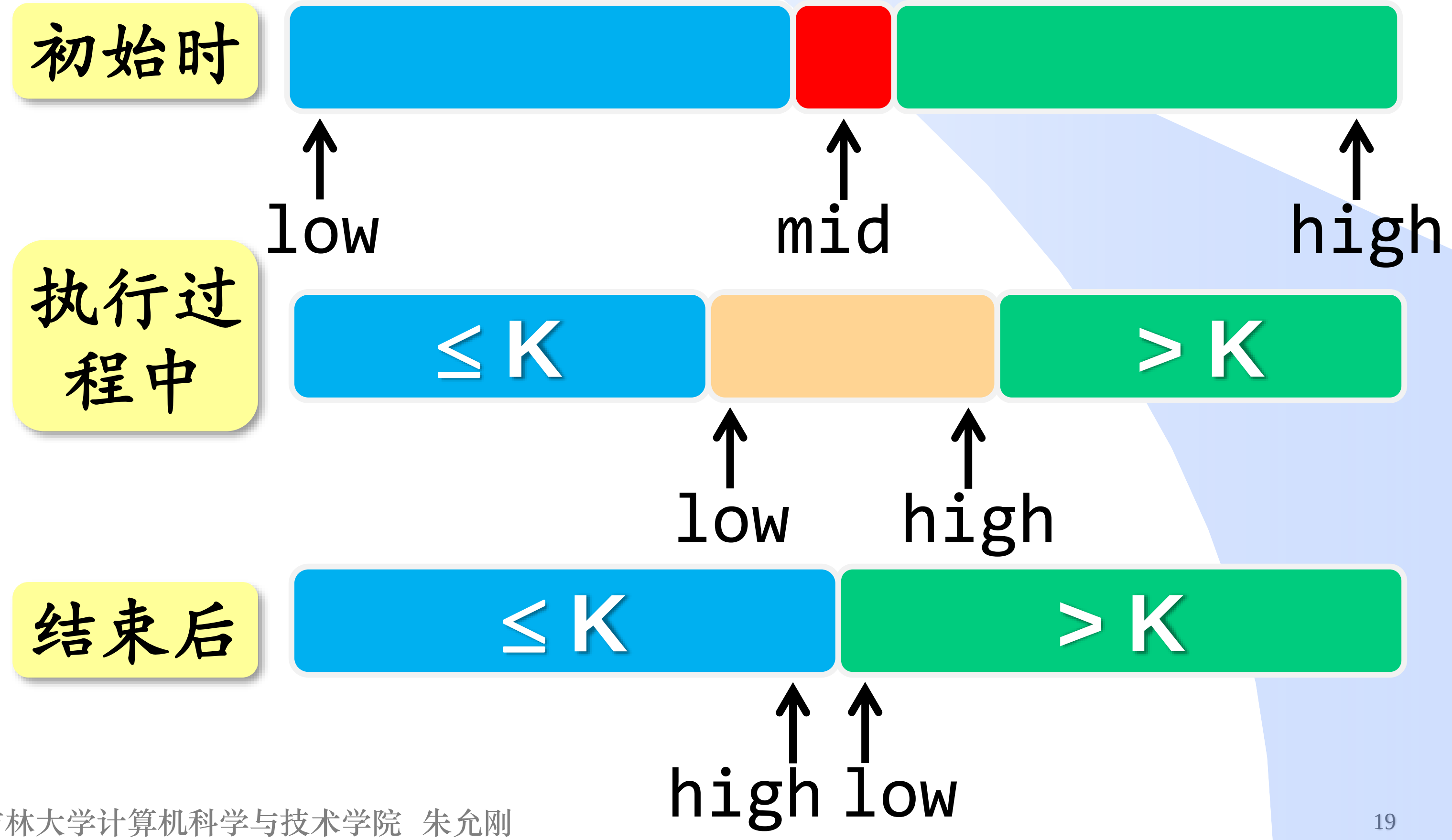
大于等于K的第一个位置

```
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

```
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

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】

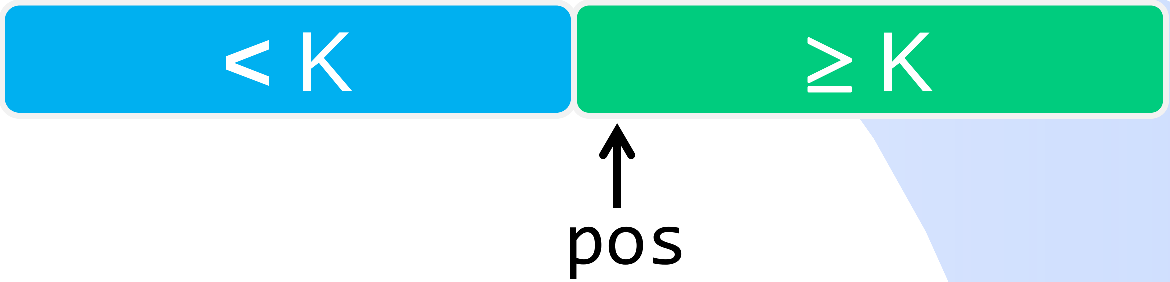
2	3	6	6	6	6	8	10
1	2	3	4	5	6	7	8

第一个等于K的位置

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*)



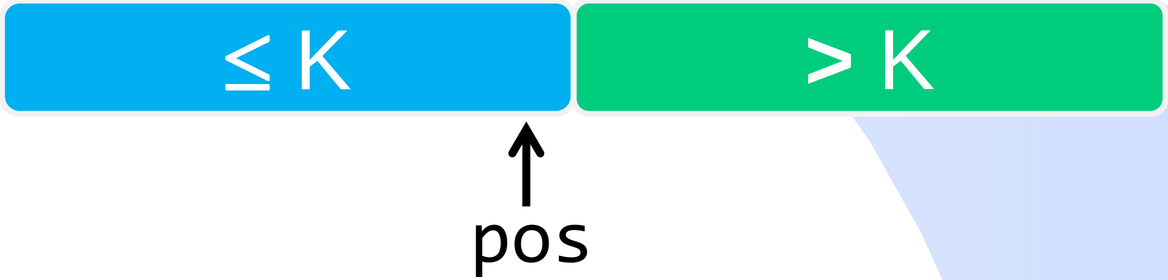

```
int Lower_bound(int R[], int low, int high, int K)
```

最后一个等于K的位置

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)
```

二分查找答案

小明喜欢吃香蕉。有n堆香蕉，第i堆中有$p[i]$根香蕉。假定他吃香蕉的速度s，即每个小时他将会选择一堆香蕉，从中吃掉s根。如果这堆香蕉少于s根，他将吃掉这堆的所有香蕉，然后这一小时内不会再吃更多的香蕉。编写程序计算他最小以什么速度吃香蕉，才能在H小时内吃掉所有香蕉。【华为、字节跳动、招商银行、中国移动、谷歌面试题LeetCode875】

p	5	3	2	10	6	7	8	9
	0	1	2	3	4	5	…	n-1

s的最小值1，s的最大值$m=\max\{p[i]\}$

常规（暴力）解法

➢ s的最小值1，s的最大值m=max{$p[i]$}

```
for(s=1; s<=m; s++){
    if(以速度s能在H时间内吃完香蕉) return s;

bool check(int p[],int n, int s, int H){
    long time = 0;
    for(int i=0; i<n; i++)
        time +=ceil(1.0*p[i]/s);
    return time<=H;
}
```

相当于有一个新的数组，下标是s，元素值是以速度s能否在H小时内吃完香蕉。暴力方案相当于从左往右扫描数组，找第一个元素值≥1的位置

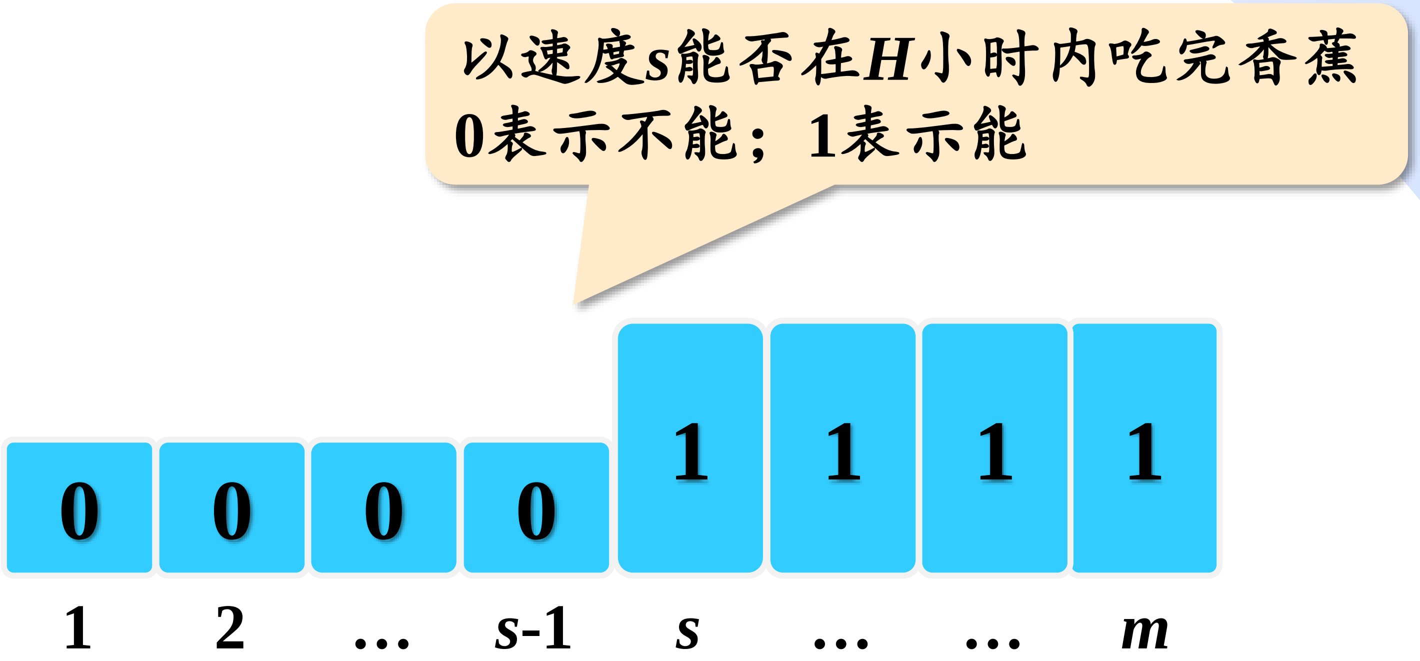
以速度s能否在H小时内吃完香蕉
0表示不能；1表示能

0	0	0	0	1	1	1	1
1	2	…	s-1	s	…	…	m

时间复杂度
O(nm)
m=max{$p[i]$}

新策略——二分查找答案

- 可通过二分查找来确定*s*。
- 找满足条件的最小*s*（在下面递增有序的数组里找元素值≥1的第一个位置）。



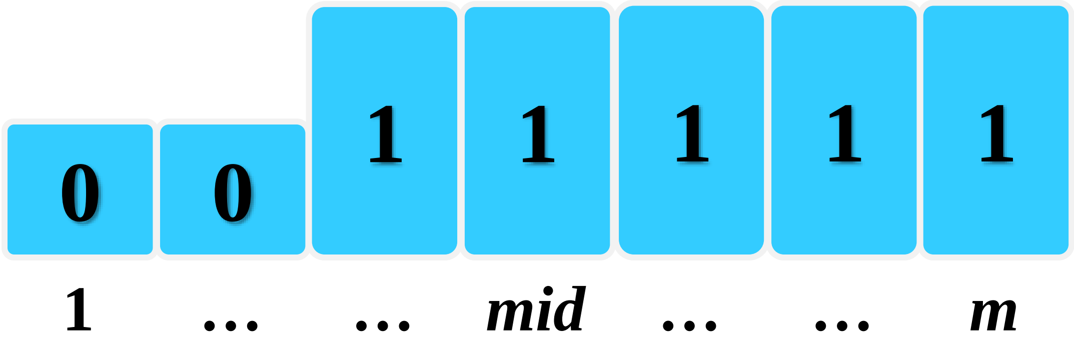
二分查找答案

```
int MinEatingSpeed(int p[], int n, int H) {
    int m=-1;  //m存max{p[i]}
    for(int i=0;i<n;i++) if(p[i]>m) m=p[i];
    int low=1, high=m;
    while(low<=high){
        int mid = (low+high)/2;
        if(check(p,n,mid,H)) high = mid-1;
        else low = mid+1;
    }
    return low;
}
```

时间复杂度

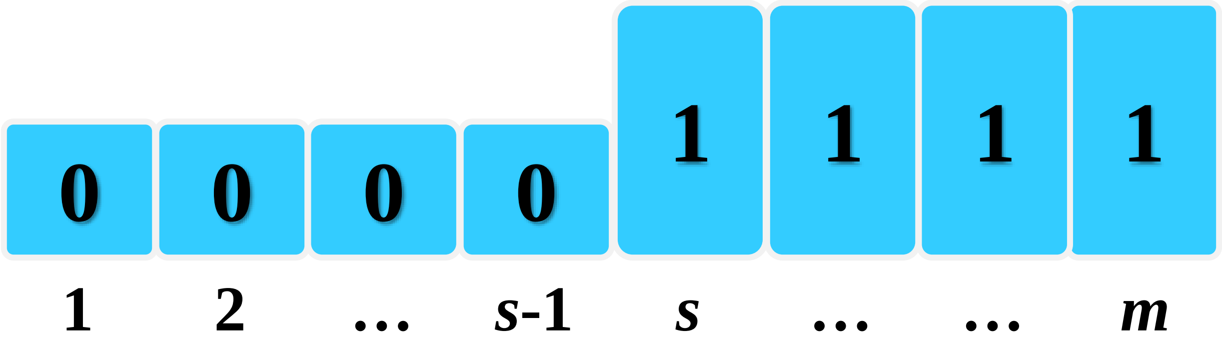
$O(n\log m)$

$m=\max\{p[i]\}$



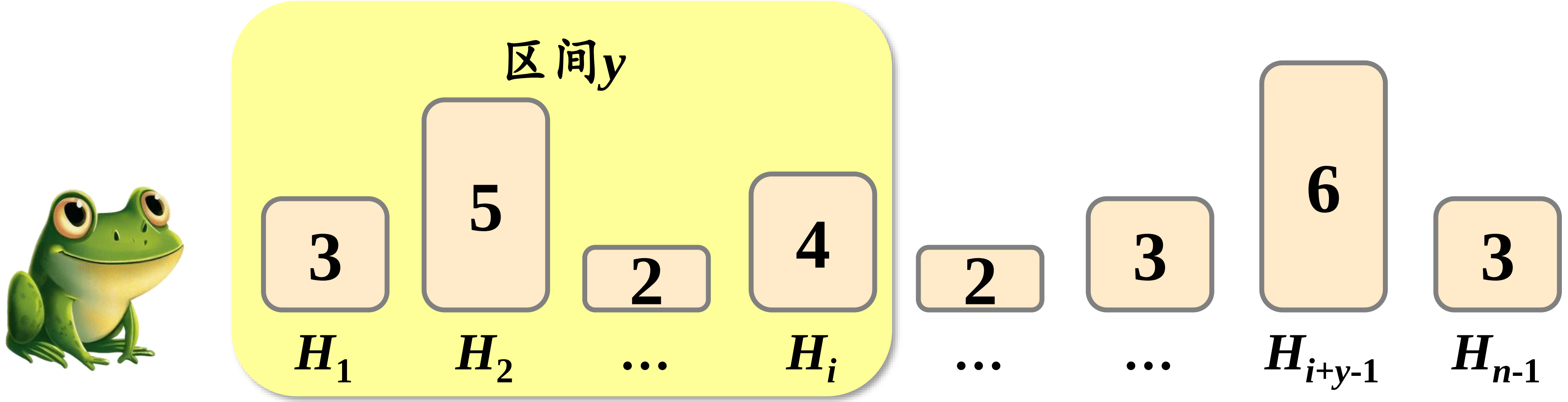
二分查找答案

- 对于一些问题，需要找某个满足条件的解。
- 首先可设计一个判定算法判断某个解是否满足条件（如上例中的check函数）。
- 把解空间的每个解都顺序枚举一遍，进而找到满足条件的解。
- 如果解空间具有单调性，则可以用“二分”替代“枚举”，快速找到最终解。



练习——青蛙过河

小青蛙住在一条河边，它想到河对岸的学校去学习，它打算经过河里的石头跳到对岸。河的长度为n，河里有n-1个石头排成一条直线，青蛙每次跳跃必须落在一块石头或者岸上。每块石头有一个高度，石头高度分别为$H_1,H_2,\ldots,H_{n-1}$，每次青蛙从一块石头起跳，该石头的高度就会下降 1，当石头高度下降到 0 时青蛙不能再跳到这块石头上。小青蛙一共需要去学校上 x 天课，所以它需要往返 $2x$ 次。当小青蛙具有一个跳跃能力 y 时，它能跳不超过 y 的距离。请问小青蛙的跳跃能力至少是多少才能用这些石头上完 x 天课。【2022年蓝桥杯省赛真题LanqiaoOJ2097】

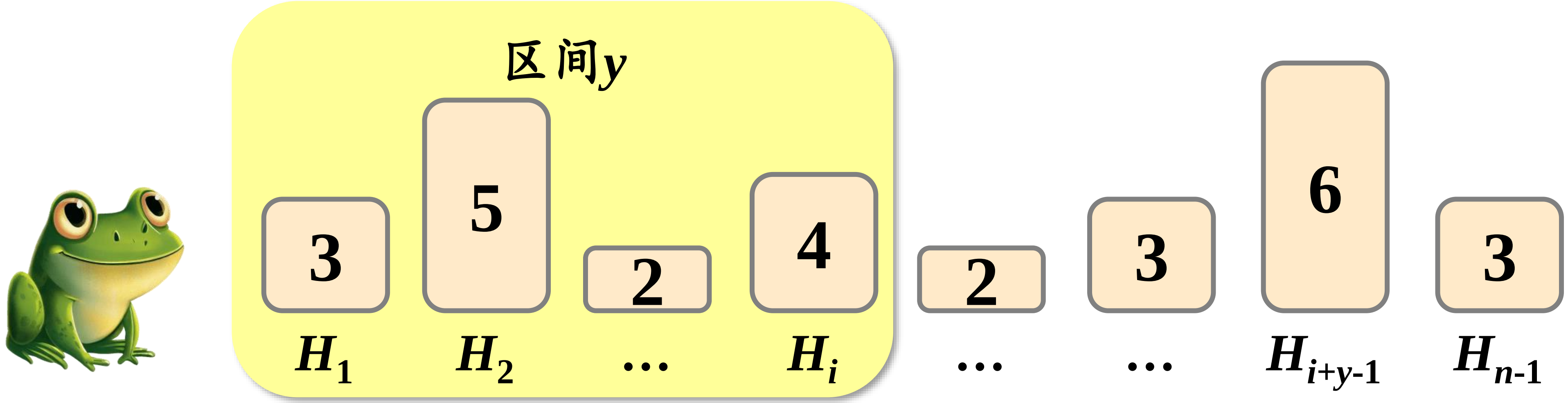


常规（暴力）解法

- y的最小值1，y的最大值n

```
for(y=1; y<=n; y++){
    if(跳跃能力y能够满足条件) return y;
```

- 1只青蛙往返$2x$次$\Leftrightarrow$ $2x$只青蛙往返1次。
- 任意区间y内，所有石头高度之和应$\geq 2x$

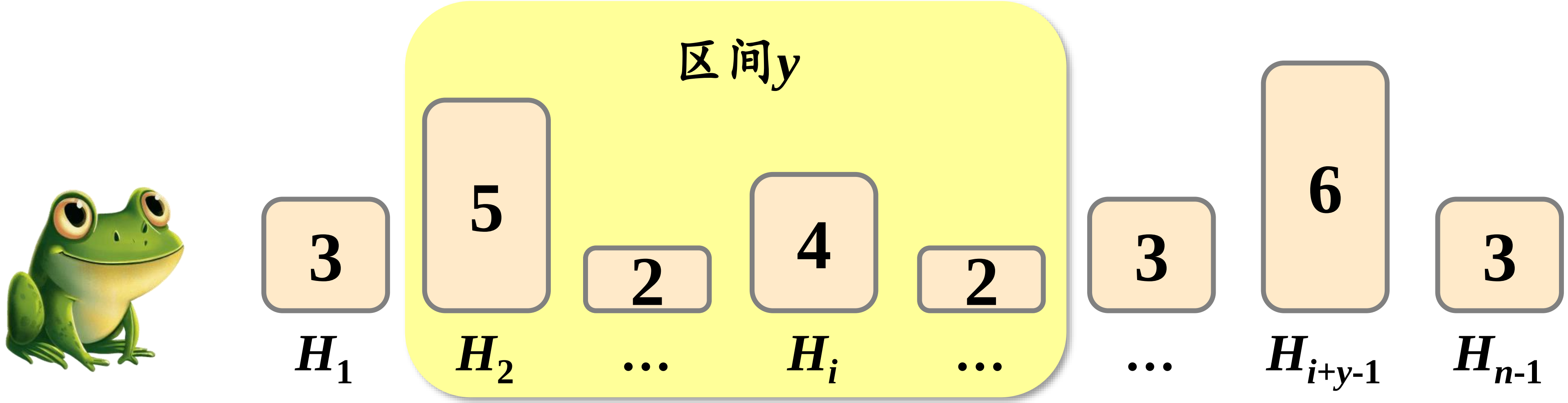


常规（暴力）解法

- y的最小值1，y的最大值n

```
for(y=1; y<=n; y++){
     if(跳跃能力y能够满足条件) return y;
```

- 1只青蛙往返$2x$次⇔ $2x$只青蛙往返1次。
- 任意区间y内，所有石头高度之和应≥ $2x$

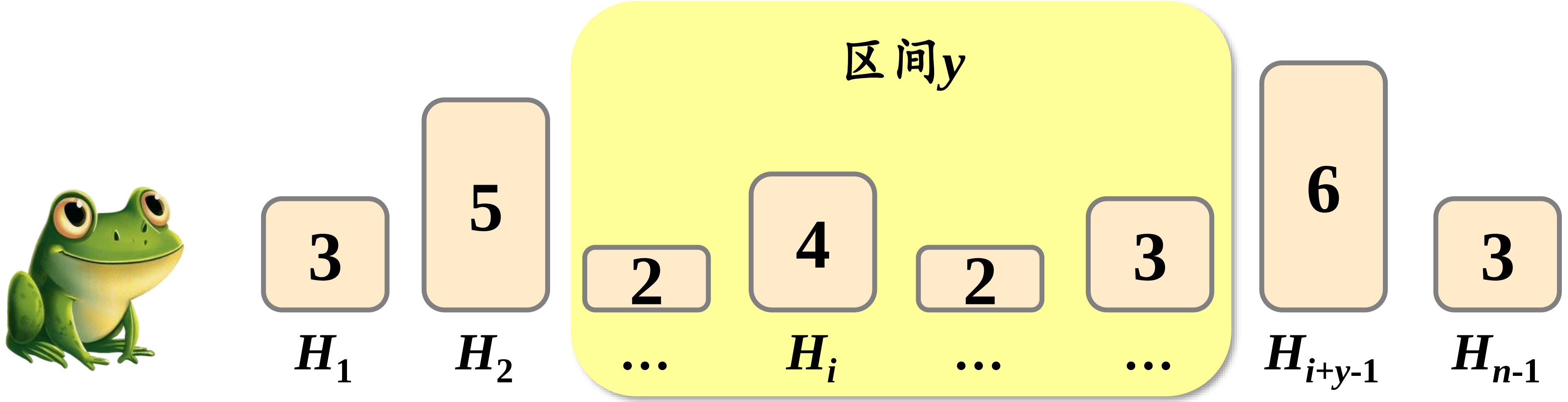


常规（暴力）解法

- y的最小值1，y的最大值n

```
for(y=1; y<=n; y++){
    if(跳跃能力y能够满足条件) return y;
```

- 1只青蛙往返$2x$次⇔ $2x$只青蛙往返1次。
- 任意区间y内，所有石头高度之和应≥ $2x$



常规（暴力）解法

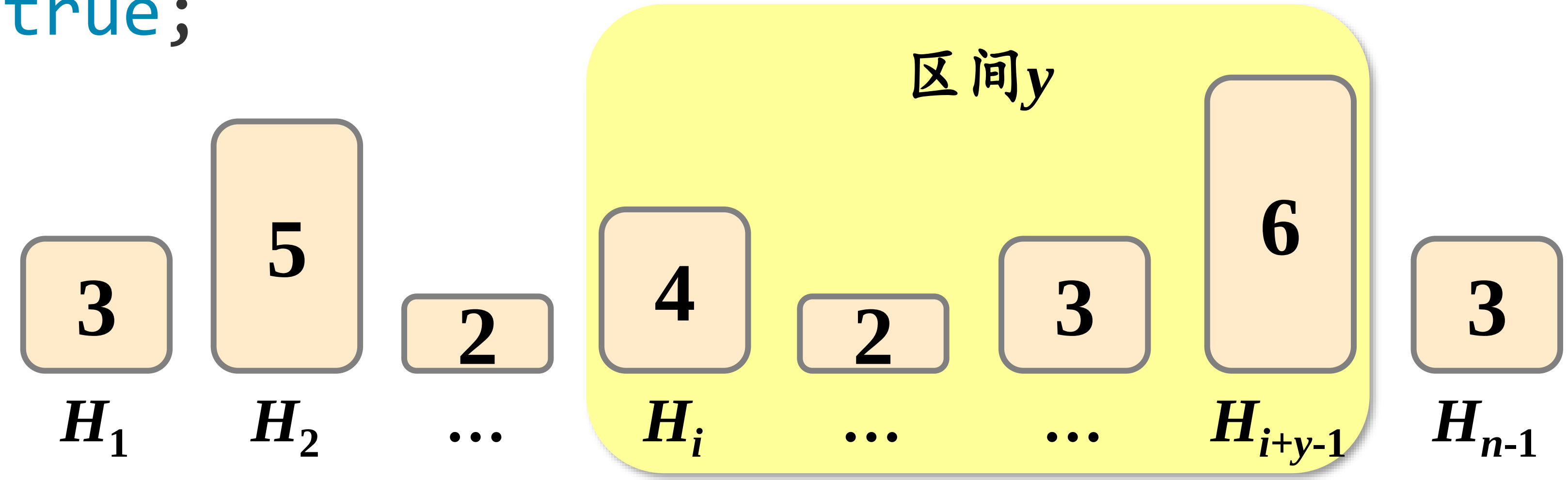
- y的最小值1，y的最大值n

```
for(y=1; y<=n; y++){
      if(跳跃能力y能够满足条件) return y;
```

- 1只青蛙往返2x次⇔ 2x只青蛙往返1次。
- 任意区间y内，所有石头高度之和应≥ 2x

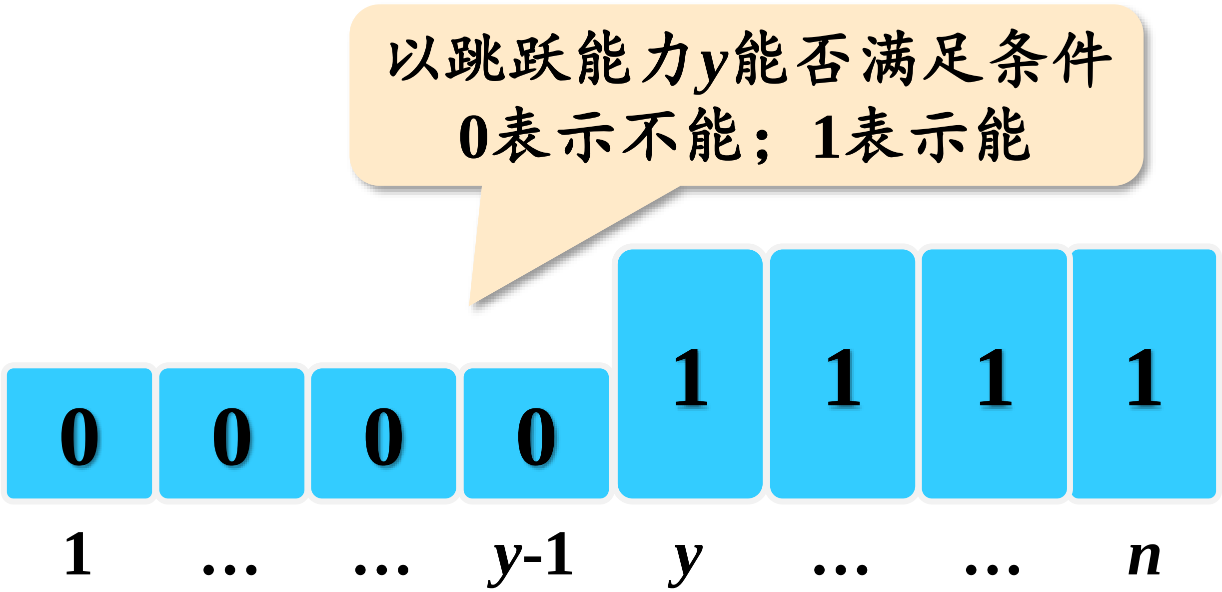
时间复杂度 $O(n^2)$

```
bool check(int H[],int n, int y, int x, int sum[]){
  for(int i=1;i<=n-y;i++) //考察以i为起点长度为y的区间Hi…Hi+y-1
    if(sum[i+y-1]-sum[i-1]<2*x) return false;
  return true;
}
```



新策略——二分法

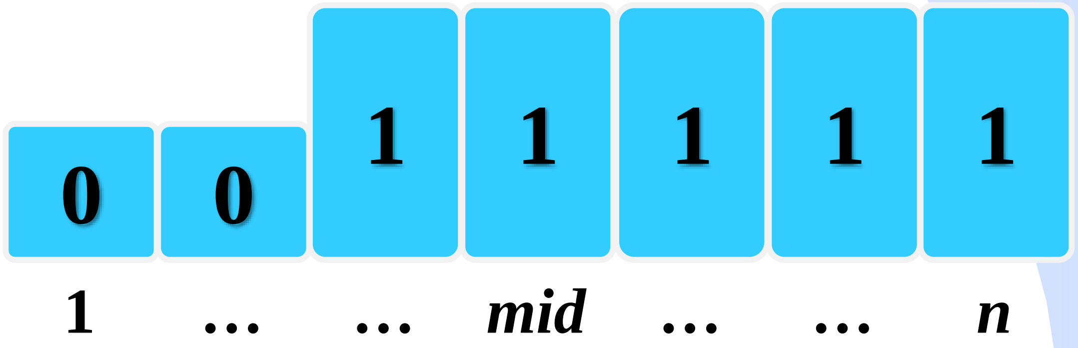
- 相当于有一个新的数组，下标是y，元素值是跳跃能力y能否满足条件。
- 暴力方案相当于从左往右扫描数组，找第一个元素值≥1的位置。
- 可通过**二分查找**来找满足条件的最小y（在下面递增有序的数组里找元素值≥1的第一个位置）。



二分法

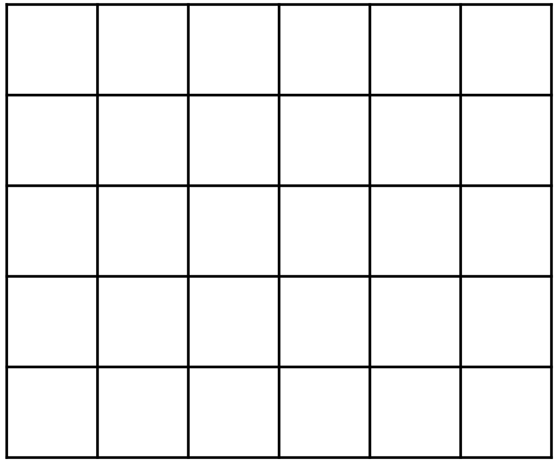
```
int MinJumpDist(int H[], int n, int x) {
    int sum[N]={0}; //存H数组的前缀和
    for(int i=1;i<=n-1;i++) sum[i]=sum[i-1]+H[i];
    int low=1, high=n;
    while(low<=high){
        int mid = (low+high)/2;
        if(check(H,n,mid,x,sum)) high = mid-1;
        else low = mid+1;
    }
    return low;
}
```

时间复杂度
O($n$log$n$)

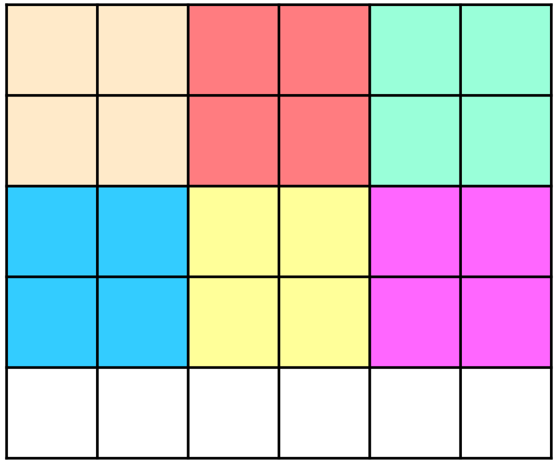


练习——分巧克力

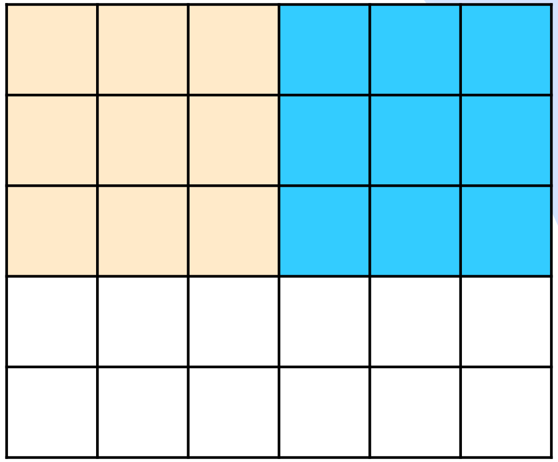
有k位小朋友到小明家做客，小明拿出巧克力招待朋友们。小明一共有n块巧克力，其中第i块是$H_i \times W_i$的方格组成的长方形。小明需要从这n块巧克力中切出k块巧克力分给朋友们。切出的巧克力需要满足（1）形状是正方形，边长是整数（2）大小相同。例如一块5×6的巧克力可以切出6块2×2的巧克力或者2块3×3的巧克力。当然小朋友们都希望得到的巧克力尽可能大，请编写程序帮助小明计算切出的巧克力的最大边长d。【2017年蓝桥杯省赛真题LanqiaoOJ99】



5×6



6块 2×2



2块 3×3

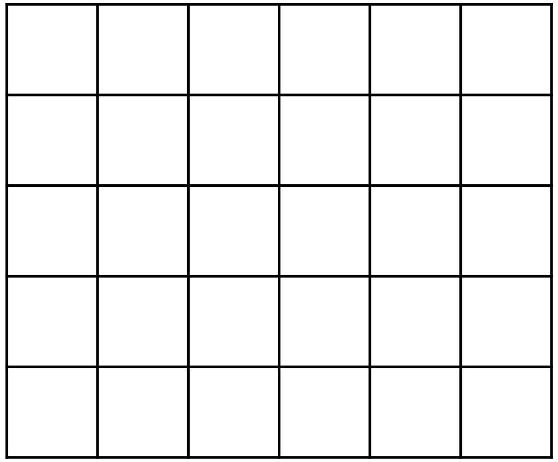
常规（暴力）解法

- d的最小值1，d的最大值m=max{H_i}

```
for(d=m; d>=1; d--){
    if(d能满足条件) return d;
```

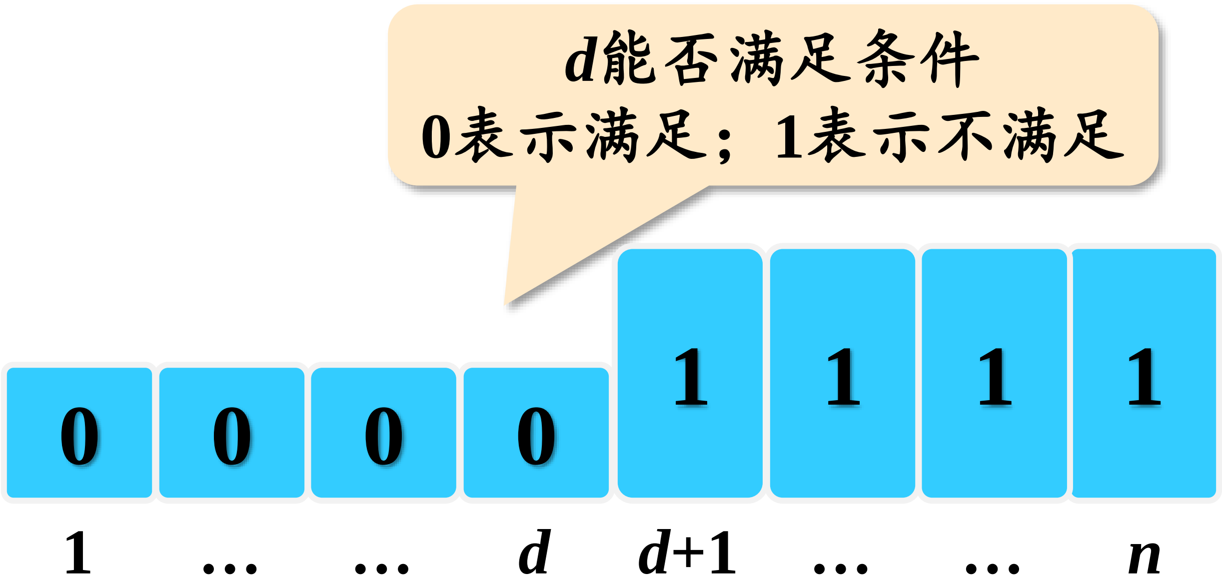
时间复杂度
O(nm)

```
bool check(int H[],int W[], int n, int k, int d){
    int sum=0;
    for(int i=0;i<n;i++)  //第i块巧克力能切出几块d×d的巧克力
        sum += (H[i]/d)*(W[i]/d);
    return sum>=k;
}
```



新策略——二分法

- 相当于有一个新的数组，下标是d，元素值是d能否满足条件（能否把n块巧克力切出至少k块$d\times d$的巧克力）。
- 暴力方案相当于顺序扫描数组，找元素值≤0的最后一个位置。
- 可通过**二分查找**来找满足条件的最后一个位置d。



二分法

```
int MaxChocolateCut(int H[], int W[], int n, int k) {
    int m=-1;  //m存max{H[i]}
    for(int i=0;i<n;i++) if(H[i]>m) m=H[i];
    int low=1, high=m;
    while(low<=high){
        int mid = (low+high)/2;
        if(check(H,W,n,k,mid)) low = mid+1;
        else high = mid-1;
    }
    return high;
}
```

时间复杂度
O(*nlogm*)