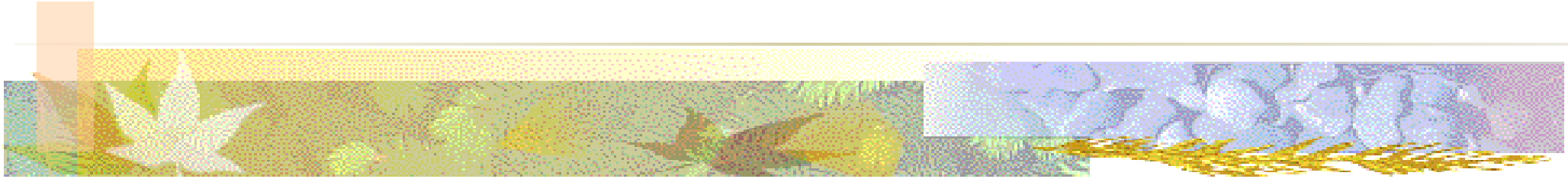
第7章 数组

——查找算法的函数实现



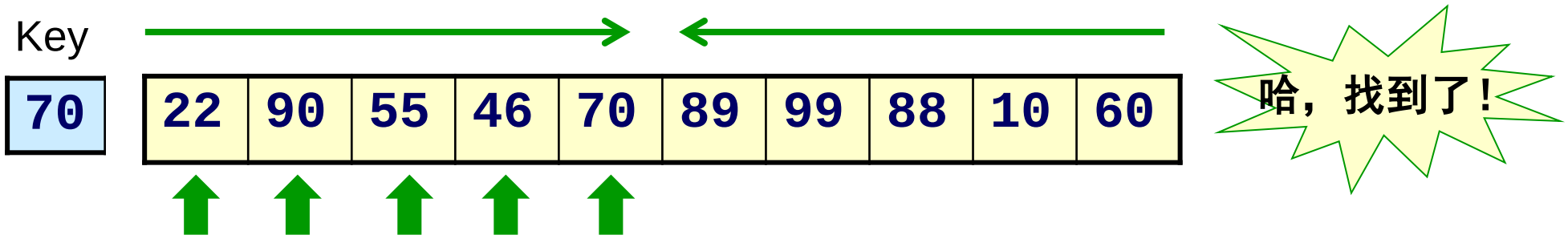
哈尔滨工业大学

苏小红

sxh@hit.edu.cn

线性查找（Linear Search）

- 不要求数据表是已排好序的
  - 从线性数据表中的第一个（或最后一个）记录开始查找
  - 依次将记录的关键字与查找关键字进行比较
    - 当某个记录的关键字与查找关键字相等时，即查找成功
    - 反之，查完全部记录都没有与之相等的关键字，则查找失败



```c
int LinSearch(long num[], long x, int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        if (num[i] == x)
        {
            return i;
        }
    }
    return -1;
}
```

找到时返回下标

找不到时返回-1

【例】查找某学号学生的成绩

```c
#define N 40
#include <stdio.h>
int ReadScore(long num[],int score[]);
int LinSearch(long num[],long x,int n);
int main()
{
  int score[N], n, pos;
  long num[N], x;
  n = ReadScore(num, score);
  printf("Input the searching ID:");
  scanf("%ld", &x);
  pos = LinSearch(num, x, n);
  if (pos != -1)
  {
    printf("score=%d\n", score[pos]);
  }
  else
  {
    printf("Not found!\n");
  }
  return 0;
}
```

```c
int ReadScore(long num[], int score[])
{
  int i = -1;
  do{
    i++;
    printf("Input num,score:");
    scanf("%ld%d", &num[i], &score[i]);
  }while (score[i] >= 0);
  return i;
}
```

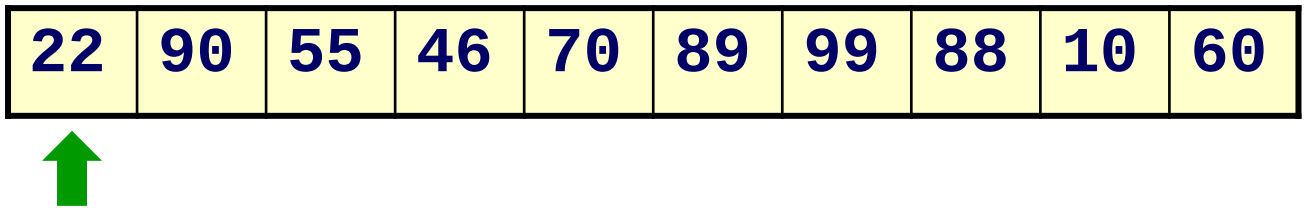
【例】查找某学号学生的成绩

```c
#define N 40
#include <stdio.h>
int ReadScore(long num[],int score[]);
int LinSearch(long num[],long x,int n);
int main()
{
  int score[N], n, pos;
  long num[N], x;
  n = ReadScore(num, score);
  printf("Input the searching ID:");
  scanf("%ld", &x);
  pos = LinSearch(num, x, n);
  if (pos != -1)
  {
    printf("score=%d\n", score[pos]);
  }
  else
  {
    printf("Not found!\n");
  }
  return 0;
}
```

线性查找（Linear Search）

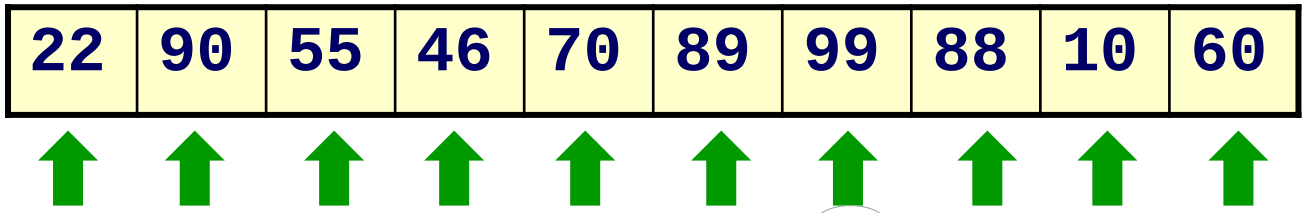
- 线性查找的性能
  - 最好情况

Key



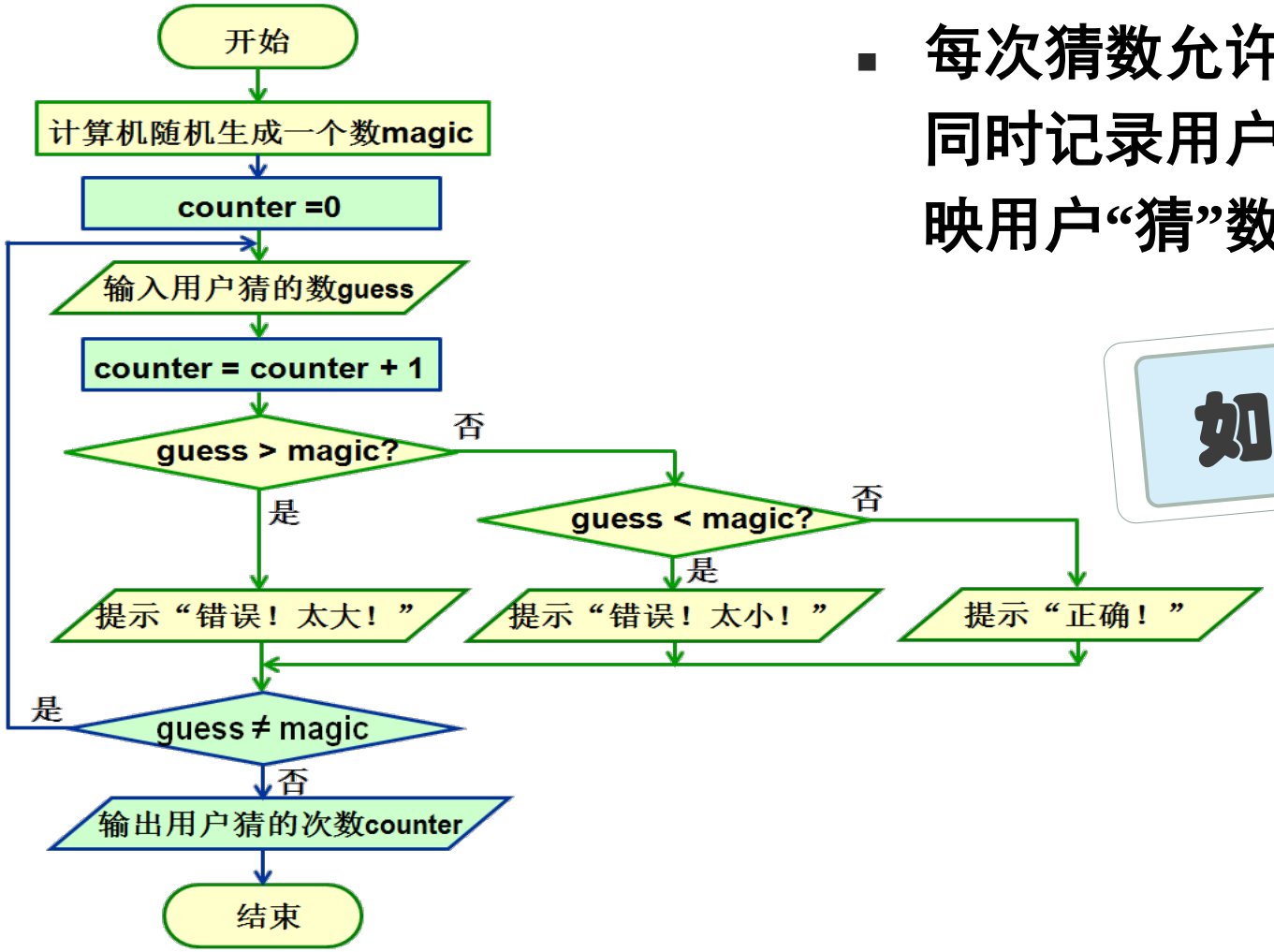
  - 最坏情况

Key

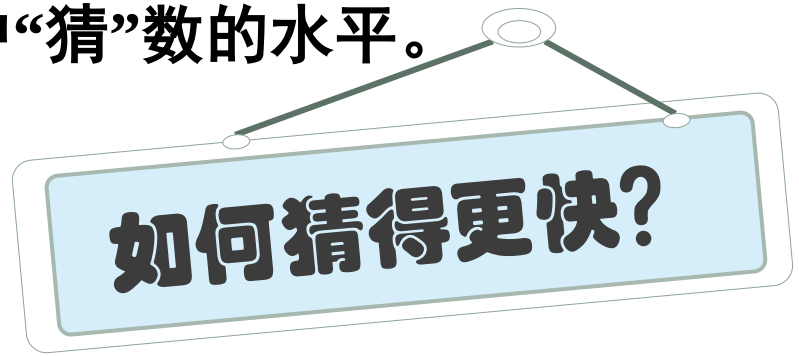


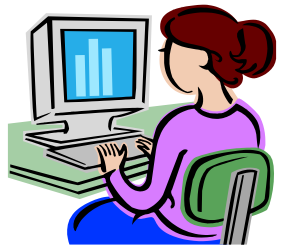
  - 平均情况
    - 查找次数是数据量的一半

猜数游戏



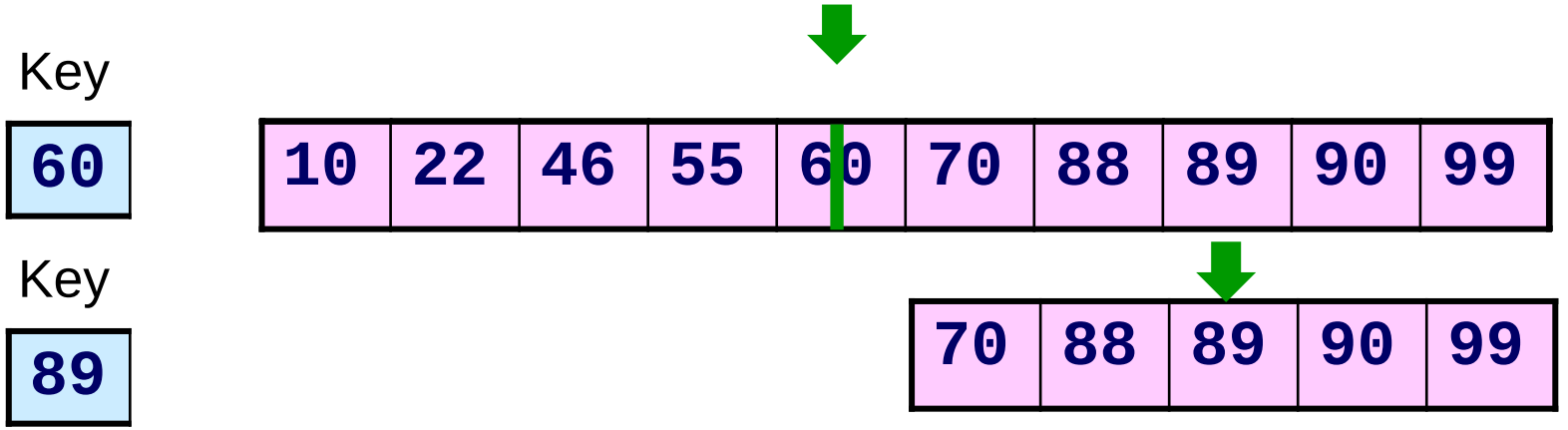
- 每次猜数允许用户直到猜对为止，同时记录用户猜的次数，以此来反映用户“猜”数的水平。





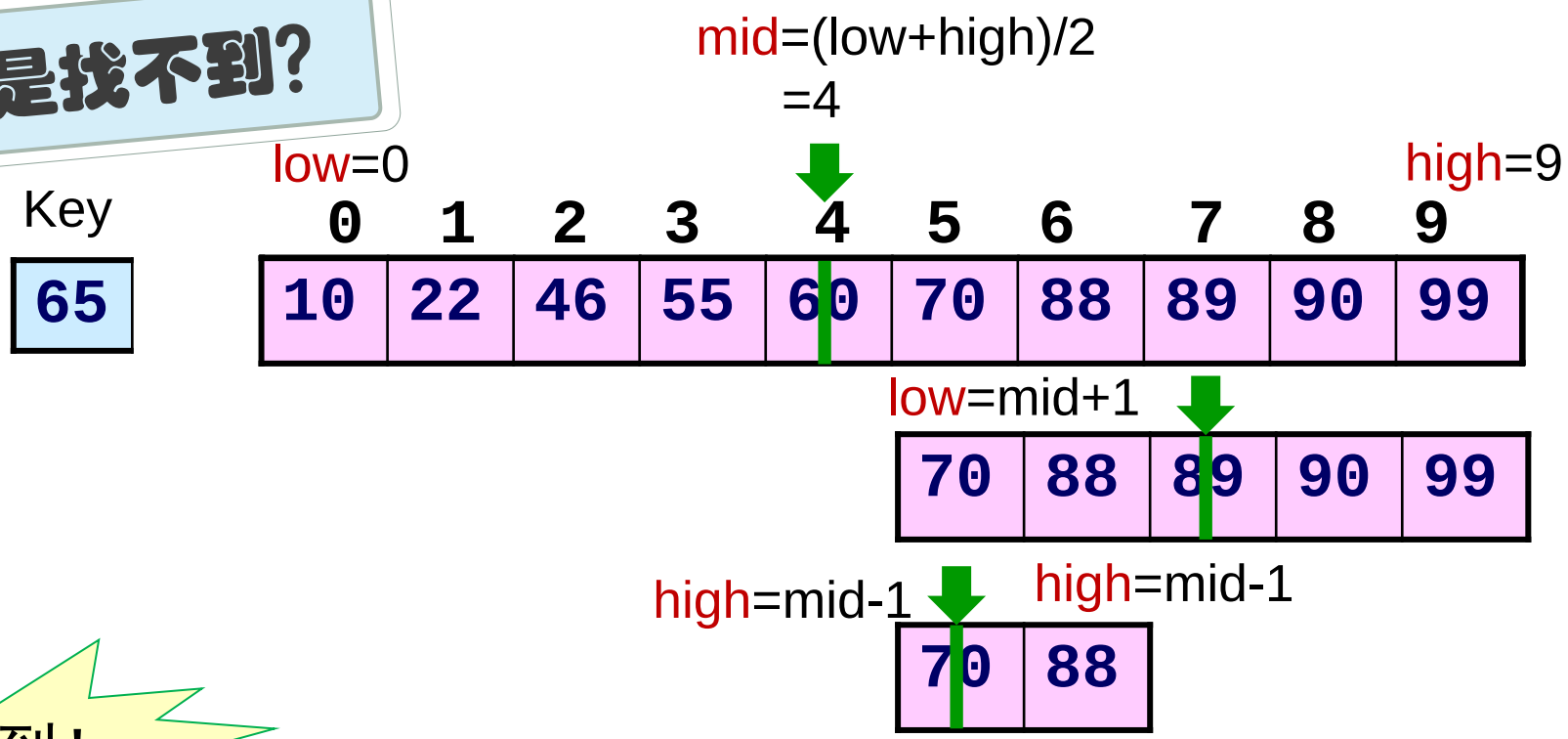
二分查找（Binary Search）

- 要求数据表是已排好序的
  - 先将表的中间位置记录的关键字与查找关键字比较
    - 如果两者相等，则查找成功
    - 否则将表分成前、后两个子表，根据比较结果，决定查找哪个子表



二分查找（Binary Search）

何时确定是找不到?



找不到！

low<=high为假

```
int  BinSearch(long num[], long x, int n)
{
   int  low = 0, high = n - 1, mid;
   while (low <= high)
   {
       mid = (high + low) / 2;
       if (x > num[mid])
       {
             low = mid + 1;
       }
       else  if (x < num[mid])
       {
             high = mid - 1;
       }
       else
       {
             return mid;
       }
   }
   return -1;
}
```

【例】折半查找学号

```
#define N 40
#include <stdio.h>
int ReadScore(long num[],int score[]);
int BinSearch(long num[],long x,int n);
int main()
{
  int score[N], n, pos;
  long num[N], x;
  n = ReadScore(num, score);
  printf("Input the searching ID:");
  scanf("%ld", &x);
  pos = BinSearch(num, x, n);
  if (pos != -1)
  {
    printf("score=%d\n", score[pos]);
  }
  else
  {
    printf("Not found!\n");
  }
  return 0;
}
```

```
int  BinSearch(long num[], long x, int n)
{
   int  low = 0, high = n - 1, mid;
   while (low <= high)
   {
       mid = (high + low) / 2;
       if (x > num[mid])
       {
             low = mid + 1;
       }
       else  if (x < num[mid])
       {
             high = mid - 1;
       }
       else
       {
             return mid;
       }
   }
   return -1;
}
```

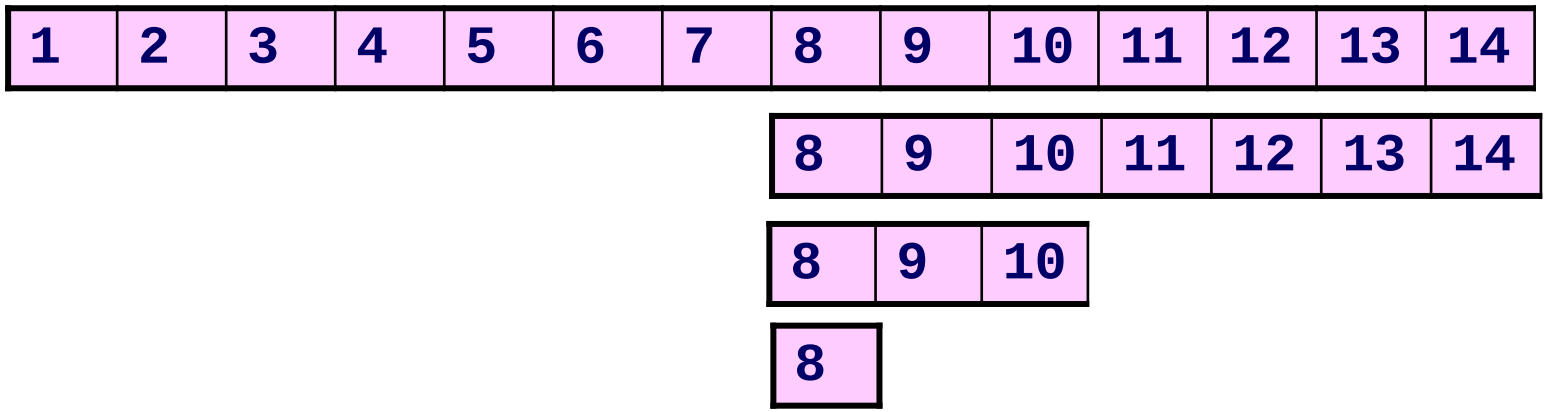
【例】折半查找学号

并非吹毛求疵，鸡蛋里挑骨头

- `mid = (high + low) / 2;`
- 如果数组很大，low和high之和大于有符号整数的极限值（在limits.h中定义）
  - 就会发生数值溢出，使mid成为一个负数
- 防止溢出的解决方案
  - 修改计算中间值的方法，用减法代替加法
  - `mid = low + (high - low) / 2;`

二分查找

- 二分查找的性能
  - 比较次数少，查找速度快，平均性能好
  - 每执行一次，都将查找空间减少一半，是计算机科学中分治思想的完美体现
  - 最多所需的比较次数是第一个大于表中元素个数的2的幂次数
    - 14（2^4 >14 ）个数，最多比较的次数是4



二分查找

- **缺点**
  - 要求待查表按关键字有序排列，否则需要先进行排序操作
  - 必须采用顺序存储结构，插入和删除数据需移动大量的数据
  - 适用于不经常变动而查找频繁的有序表

讨论

- 查找算法与我们的生活密切相关，说说你在生活中经常使用的查找操作？举例说明什么情况适合线性查找，什么情况适合二分查找？
- 二分查找还可用于计算方程的根、计算两条线段的交点等，开动你的脑筋，想想它还可用在哪里？


Thank you for
your attention !
SuXiaoHong


Q&A