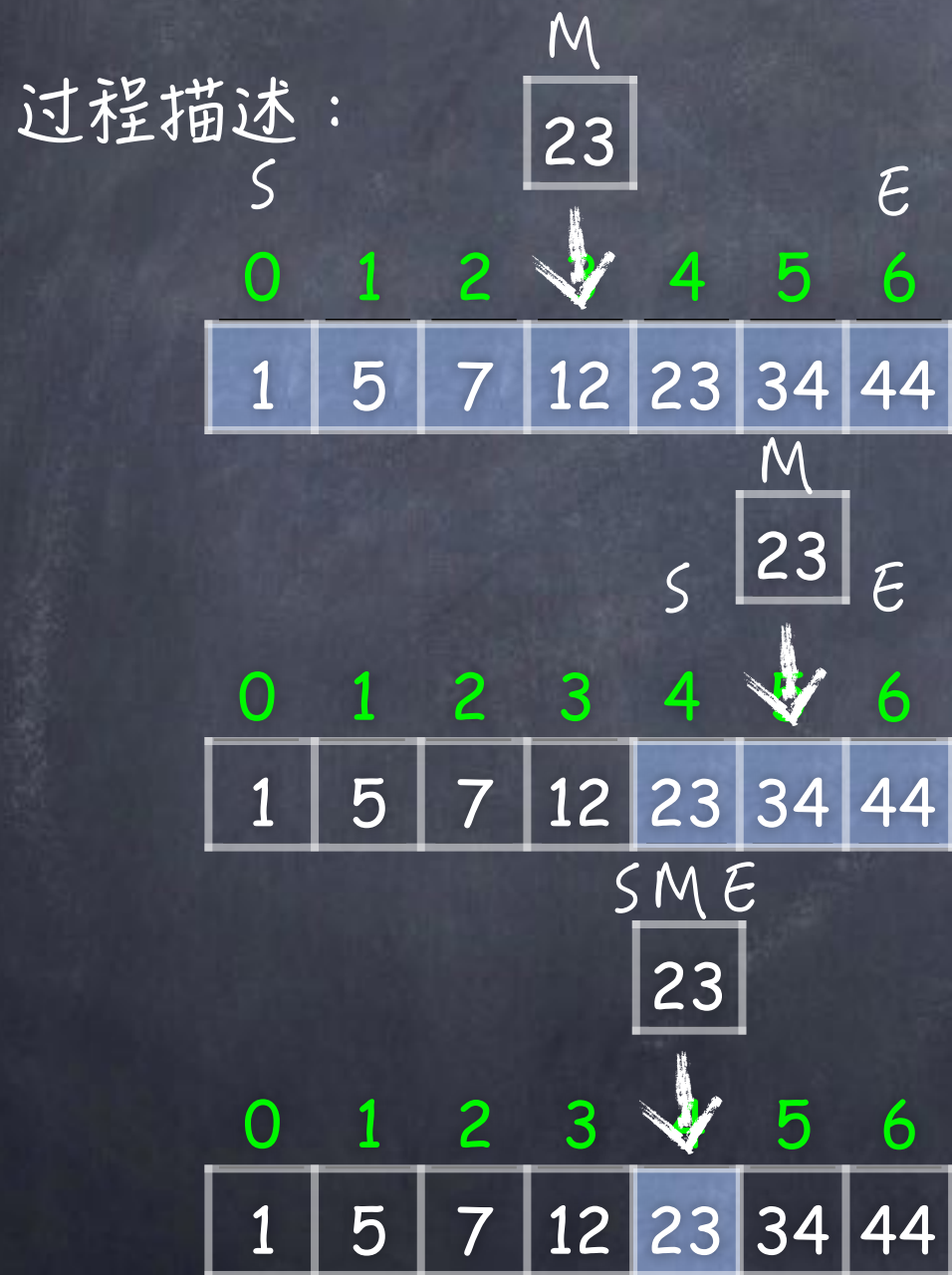


二分查找

前提： 1. 数据存放在数组中，保证提取数据为 $O(1)$ ，链表不行
2. 数据有序



1. 假设在有序数组中查找23的位置
2. 每次对半分为两个区间S-M和M-E, 即 $M=(S+E)/2$
3. 如果 $23==a[M]$, 即找到
4. 如果 $23<a[M]$, 则说明23在S-M区间, 则下次查找 $E=M-1$
5. 如果 $23>a[M]$, 则说明23在M-E区间, 则下次查找 $S=M+1$

第1次查找： $S=0, E=6, M=(S+E)/2=3$, 由于 $23>12$, 所以下次在M-E区间查找

第2次查找： $S=4, E=6, M=(S+E)/2=5$, 由于 $23<34$, 所以下次在S-M区间查找

第3次查找： $S=4, E=4, M=(S+E)/2=4$, 由于 $23=a[M]$, 所以返回查找结果

Python递归实现：

```
# -*- coding: utf-8 -*-
def BinarySearch_rec(array,item):
    length = len(array)
    if length == 0: #当长度为0时, 即没有找到
        return False
    mid = length / 2 #每次对半
    if array[mid] == item:
        return True
    elif array[mid] > item:
        return BinarySearch_rec(array[:mid],item) #前半数组
    else:
        return BinarySearch_rec(array[mid+1:],item) #后半数组

if __name__ == '__main__':
    L = [1, 5, 7, 12, 23, 34, 44]
    item = 23
    print BinarySearch_rec(L,item)
```

时间复杂度：由递归公式有
假设 $T(1) = 1$, c 为每次调用常量
 $T(n) = T(n/2) + c$

$$= T(n/4) + c + c$$
$$\dots$$

$$= T(1) + kc$$

由 $n = 2^k$, 得 $k = \log n$, 代回

$$T(n) = 1 + c \log n$$

故时间复杂度为： $O(\log n)$

Python非递归实现：

```
# -*- coding: utf-8 -*-
def BinarySearch(array, item):
    length = len(array)
    start = 0
    end = length - 1

    while start <= end:
        mid = (start + end) / 2 # 计算每次mid位置
        if array[mid] == item:
            return True
        elif item < array[mid]:
            end = mid - 1 # 前半区间S-M
        else:
            start = mid + 1 # 后半区间M-E
    return False

if __name__ == '__main__':
    L = [1, 5, 7, 12, 23, 34, 44]
    item = 23
    print "非递归实现结果：" , BinarySearch(L, item)
```

时间复杂度：由于每次都是对半查找，那么所需要的查找次数 $n/2, n/4 \cdots n/(2^k)$, $k = \log n$
故时间复杂度为： $O(\log n)$