# 本小节内容

**2014 年 42 题真题讲解**

## 2014 年 42 题真题讲解

42. （15 分）一个长度为 $L$（$L \geq 1$）的升序序列 S，处在第 $\lfloor L/2 \rceil$ 个位置的数称为 S 的中位数。例如，若序列 S1 = (11, 13, 15, 17, 19)，则 S1 的中位数是 15，两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若 S2 = (2, 4, 6, 8, 20)，则 S1 和 S2 的中位数是 11。现在有两个等长升序序列 A 和 B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列 A 和 B 的中位数。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C、C++或 Java 语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

**答案解析：**

这个题目所考察的内容是二分查找，但是有两个数组，是双数组的二分查找，是一道非常经典的题目。因为空间尽可能高效，因此我们不能够去再搞一个大数组，把两个数组合并到一起，这样得分会非常低。

（1）算法的基本设计思想如下：

分别求出序列 A 和 B 的中位数，设为 a 和 b，求序列 A 和 B 的中位数过程如下：

1）若 a=b，则 a 或 b 即为所求中位数，算法结束。

2）若 a<b，则舍弃序列 A 中较小的一半，同时舍弃序列 B 中较大的一半,要求舍弃的长度相等；

3）若 a>b，则舍弃序列 A 中较大的一半，同时舍弃序列 B 中较小的一半，要求舍弃的长度相等；

在保留的两个升序序列中，重复过程 1）、2）、3），直到两个序列中均只含一个元素时为 止，较小者即为所求的中位数。

（2）代码实现如下：

```
#include <stdio.h>
int MidSearch(int A[], int B[], int n) {
    //分别表示序列 A 和 B 的首位数、末位数和中位数，s 是 start 简写，d 是 end 简写
    int s1 = 0, d1 = n - 1, m1, s2 = 0, d2 = n - 1, m2;
    //循环判断结束条件是，两个数组均不断删除最后均只能剩余一个元素
    while (s1 != d1 || s2 != d2) {
        m1 = (s1 + d1) / 2;
        m2 = (s2 + d2) / 2;
        if (A[m1] == B[m2])
            return A[m1]; //满足条件 1）
        if (A[m1] < B[m2]) { //满足条件 2)
```

```
            if ((s1 + d1) % 2 == 0) { //若元素个数为奇数,这里注意数组下标从 0 开始
                s1 = m1; //舍弃 A 中间点以前的部分且保留中间点
                d2 = m2; //舍弃 B 中间点以后的部分且保留中间点
            }
            else { //元素个数为偶数
                s1 = m1 + 1; //舍弃 A 中间点及中间点以前部分，我们举一个例子告诉大家必须
加 1
                d2 = m2; //舍弃 B 中间点以后部分且保留中间点
            }
        }
        else { //满足条件 3），下面的操作和上面条件 2 是完全对称的
            if ((s1 + d1) % 2 == 0) { //若元素个数为奇数
                d1 = m1; //舍弃 A 中间点以后的部分且保留中间点
                s2 = m2; //舍弃 B 中间点以前的部分且保留中间点
            }
            else { //元素个数为偶数
                d1 = m1; //舍弃 A 中间点以后部分且保留中间点
                s2 = m2+1; //舍弃 B 中间点及中间点以前部分
            }
        }
    }
    return A[s1] < B[s2] ? A[s1] : B[s2];//因为题目要的是 11，因此我们拿小的那个
}

int main()
{
    int A[] = { 11,13,15,17,19};//我们也可以分别把 A 和 B 都变为偶数个元素来测试
    int B[] = { 2,4,6,8,20};
    int mid = MidSearch(A, B, 5);
    printf("mid=%d\n", mid);
    return 0;
}
```

（3）算法的时间复杂度为 $O(\log_2 n)$，空间复杂度为 $O(1)$。

因为我们没有使用额外的跟 n 相关的空间，因为不断的二分，次数是 $\log_2 n$，所以时间复杂度是 $O(\log_2 n)$