01-E-2 减而治之 & 01-E-3 递归跟踪 递归方程

#数据结构邓神

减而治之

```
Decrease-and-conquer
```

当我们面临一个较大的原始问题：

我们可以对他进行分解：

分解一个规模缩减的字问题和一个平凡的问题

我们可以先解决这个平凡的问题

然后再去分解那个规模缩减的问题为一个规模又缩减的问题和另外一个平凡的问题

如此循环 原问题就能被分为很多个平凡的问题，从而被处理

数组求和：线性递归

```
int sum(int A[],int n){
    return (n < 1) ? 0 : sum(A,n-1) + A[n-1];
}
```

这个就是减而治之的例子

将 N 个整数求和问题 编程 求 前 N-1 个整数 再加上最后一个整数的问题

其中求前 N-1 个整数的和是一个规模缩减的问题，而加上第 N 个整数是一个平凡的问题

当然这个算法效率相对不高

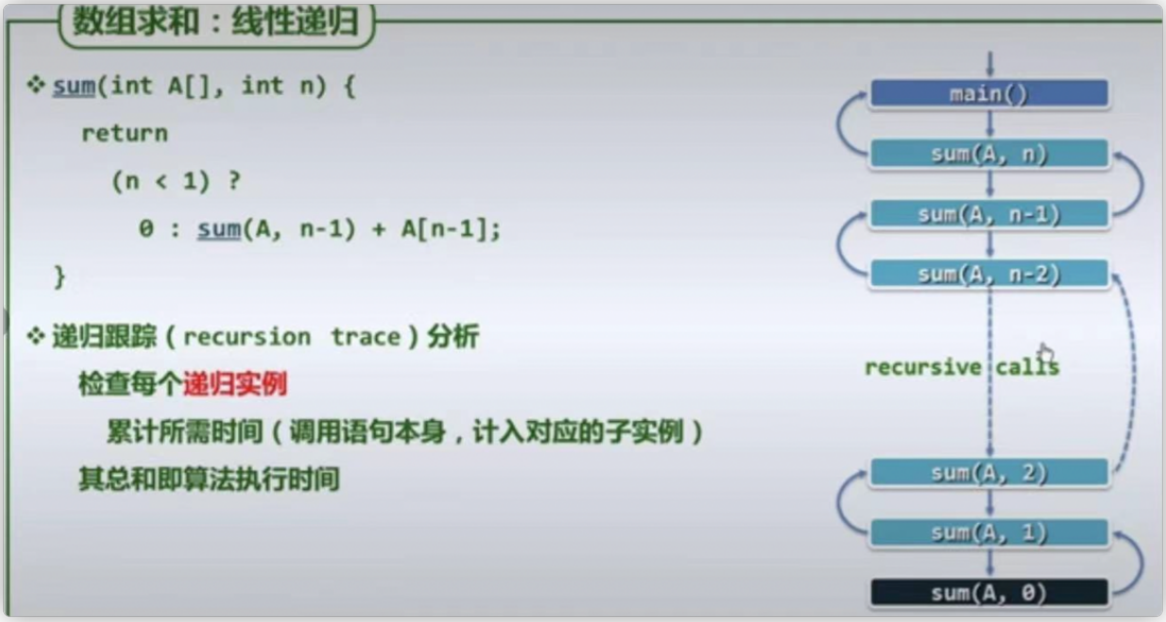
Q: 这个算法的复杂度是多少呢?

递归跟踪 (recursion trace)分析

检查每一个递归实例

累计所需时间（调用语句本身，计入对应子实例）

其总和即为算法执行时间



为了求解规模 N 的问题 会生成一个规模为 N - 1的问题 ，一直分解下去直到规模为 0

将 调用方的 sum(A,n-1) 时间抹去，算到被调用着里面去

简单计算我们发现所有调用的递归实例的计算时间都为 O(1)

一共调用了 N 次递归 时间复杂度还是 O(n)

T(n) = O(1) * (n+1) = O(n) // main函数也算一次 O(1)

不难看出，递归跟踪这种计算方法应用的范围还是很有限的，如果递归比较复杂，就很分析

递推方程

从递推的角度看，为了求解 sum(A,n)

需要求解规模为 n-1 的问题 sum(A,n-1) // T(n-1)

再累加上 A[n-1] // O(1)

递归基: sum(A, 0) // O(1)

递归方程:

T(n) = T(n-1) + O(1) // recurrence 递推表达式

T(0) = O(1) // base

这种求解有点像求解微分方程!

我们还不知道而且需要去求解 T(n) 的显式形式，但是我们知道 T(n) 的递归关系式子

我们可以对 T(n-1) 进行迭代就可以获得方程的解