01-E1

绪论

迭代与递归：减而治之

虽我之死，有子存焉；子又生孙，孙又生子；子又有子，子又有孙；子子孙孙无穷匮也，而山不加增，何苦而不平？

邓俊辉
deng@tsinghua.edu.cn

Sum

❖ 问题：计算任意n个整数之和

❖
```
int SumI( int A[], int n ) {
    int sum = 0; //O(1)
    for ( int i = 0; i < n; i++ )//O(n)
        sum += A[i]; //O(1)
    return sum; //O(1)
}
```

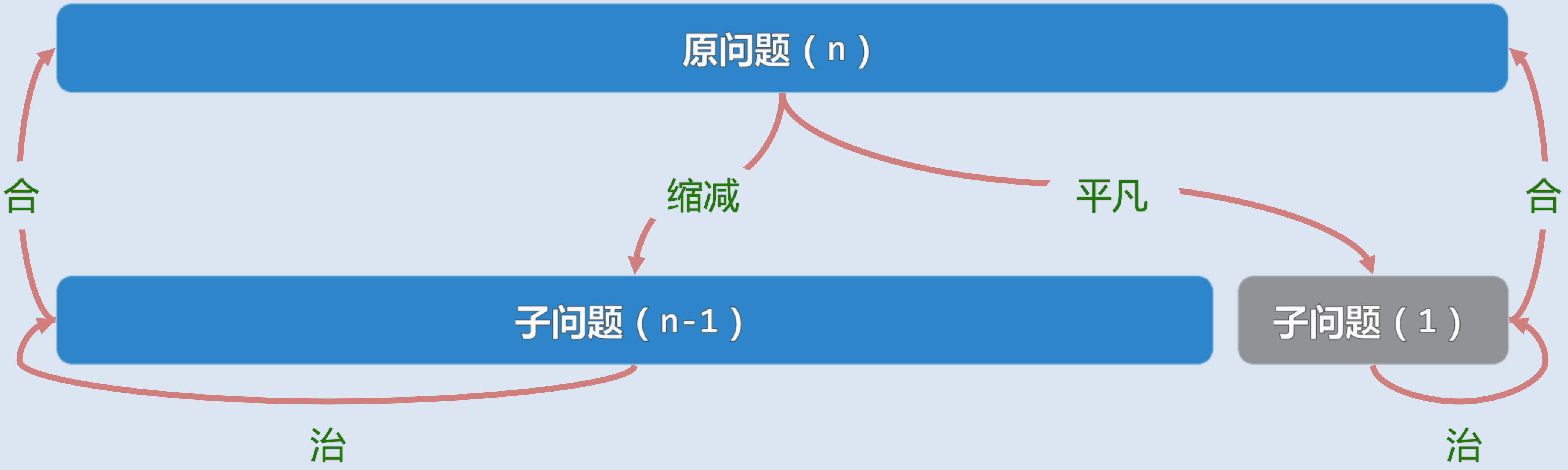
❖ 思路：逐一取出每个元素，累加之

❖ 无论A[]内容如何，都有：

$$
\begin{aligned}
\mathrm{T}(\mathrm{n}) &= 1 + \mathrm{n} \cdot 1 + 1 \\
&= n + 2 \\
&= \mathcal{O}(n) \\
&= \Omega(n) \\
&= \Theta(n)
\end{aligned}
$$

❖ 空间呢？

Decrease-and-conquer



❖ **为求解一个大规模的问题，可以**

- **将其划分为两个子问题：其一平凡，另一规模缩减** **//单调性**

- **分别求解子问题；再由子问题的解，得到原问题的解**

Linear Recursion: Trace

❖ sum(int A[], int n)

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

❖ 递归跟踪：绘出计算过程中出现过的所有**递归实例**（及其调用关系）

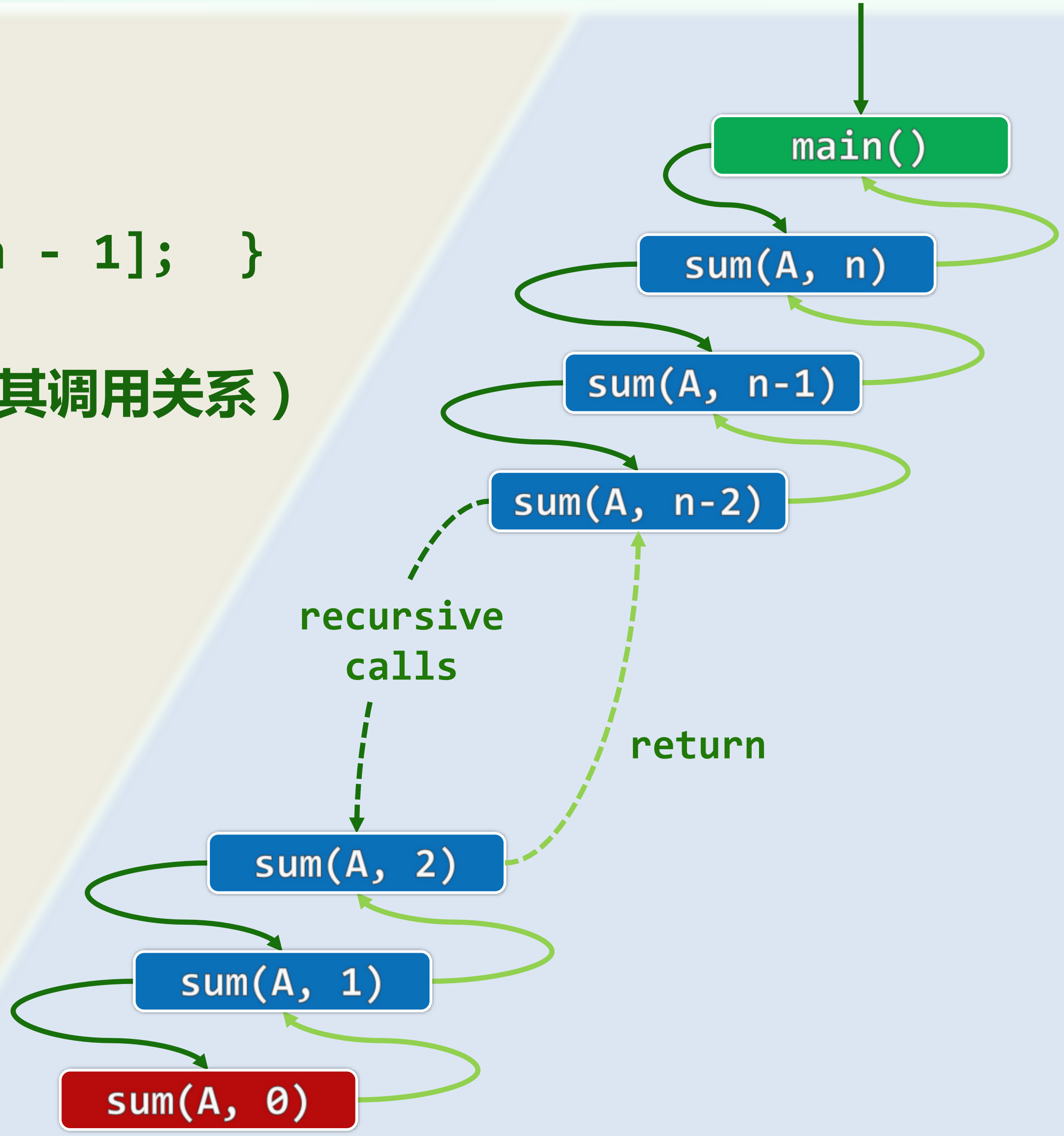
- 它们**各自**所需时间之总和，即为整体运行时间

- （**调用**操作本身的成本，如何处理？）

❖ 本例中，共计n+1个递归实例，各自只需$\mathcal{O}(1)$时间

故总体运行时间为：$T(n) = \mathcal{O}(1) \times (n+1) = \mathcal{O}(n)$

❖ 空间复杂度呢？



Linear Recursion: Recurrence

- 对于大规模的问题、复杂的递归算法，递归跟踪不再适用

  此时可采用另一抽象的方法...

- 从递推的角度看，为求解规模为n的问题sum(A, n)，需 //T(n)

  - 递归求解规模为n-1的问题sum(A, n - 1)，再 //T(n-1)

  - 累加上A[n - 1] //$\mathcal{O}(1)$

- 递推方程：$T(n) = T(n-1) + \mathcal{O}(1)$ //recurrence

  $T(0) = \mathcal{O}(1)$ //base: sum(A, 0)

- 求解：$T(n) = T(n-2) + \mathcal{O}(2) = T(n-3) + \mathcal{O}(3) = \ldots = T(0) + \mathcal{O}(n) = \mathcal{O}(n)$

Reverse

❖ void reverse(int * A, int lo, int hi);

//将数组中的区间A[lo,hi]前后颠倒

❖ 减治：$Rev(lo, hi) \ = \ [hi] \ + \ Rev(lo+1, hi-1) \ + \ [lo]$

```
❖ if (lo < hi) { //递归版
      swap( A[lo], A[hi] );
      reverse( A, lo + 1, hi - 1 );
  } //线性递归（尾递归），O(n)
❖ while (lo < hi) //迭代版
      swap( A[lo++], A[hi--] ); //亦是O(n)
```