

分治 (Divide And Conquer)

@M了个J

<https://github.com/CoderMJLee>

<http://cnblogs.com/mjios>

码拉松



实力IT教育 www.520it.com

分治 (Divide And Conquer)

■ 分治，也就是分而治之。它的一般步骤是

- ① 将原问题分解成若干个规模较小的子问题（子问题和原问题的结构一样，只是规模不一样）
- ② 子问题又不断分解成规模更小的子问题，直到不能再分解（直到可以轻易计算出子问题的解）
- ③ 利用子问题的解推导出原问题的解

■ 因此，分治策略非常适合用递归

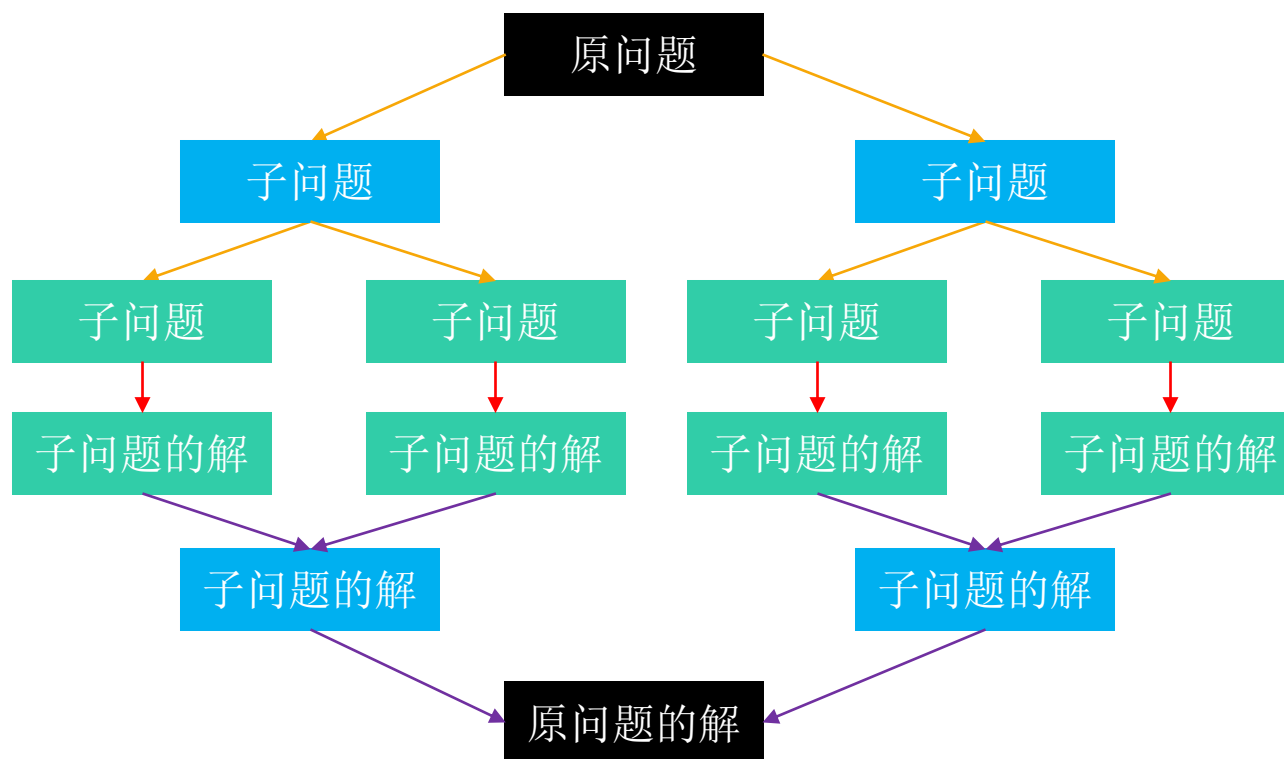
■ 需要注意的是：子问题之间是相互独立的

■ 分治的应用

□ 快速排序

□ 归并排序

□ Karatsuba算法（大数乘法）



主定理 (Master Theorem)

- 分治策略通常遵守一种通用模式
- 解决规模为 n 的问题，分解成 a 个规模为 $\frac{n}{b}$ 的子问题，然后在 $O(n^d)$ 时间内将子问题的解合并起来
- 算法运行时间为： $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$, $a > 0, b > 1, d \geq 0$
 - ✓ $d > \log_b a$, $T(n) = O(n^d)$
 - ✓ $d = \log_b a$, $T(n) = O(n^d \log n)$
 - ✓ $d < \log_b a$, $T(n) = O(n^{\log_b a})$
- 比如归并排序的运行时间是： $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, $a = 2, b = 2, d = 1$, 所以 $T(n) = O(n \log n)$
- 思考：为什么有些问题采取分治策略后，性能会有所提升？

练习1 – 最大连续子序列和

■ leetcode_53_最大子序和: <https://leetcode-cn.com/problems/maximum-subarray/>

■ 给定一个长度为 n 的整数序列，求它的最大连续子序列和

□ 比如 -2 、 1 、 -3 、 4 、 -1 、 2 、 1 、 -5 、 4 的最大连续子序列和是 $4 + (-1) + 2 + 1 = 6$

■ 这道题也属于最大切片问题（最大区段，Greatest Slice）

■ 概念区分

□ 子串、子数组、子区间必须是连续的，子序列是可以不连续的

解法1 – 暴力出奇迹

- 穷举出所有可能的连续子序列，并计算出它们的和，最后取它们中的最大值

```
int maxSubArray(int[] nums) {  
    if (nums == null || nums.length == 0) return 0;  
    int max = Integer.MIN_VALUE;  
    for (int begin = 0; begin < nums.length; begin++) {  
        for (int end = begin; end < nums.length; end++) {  
            int sum = 0;  
            for (int i = begin; i <= end; i++) {  
                sum += nums[i];  
            }  
            max = Math.max(max, sum);  
        }  
    }  
    return max;  
}
```

- 空间复杂度: $O(1)$, 时间复杂度: $O(n^3)$

解法1 – 暴力出奇迹 – 优化

■ 重复利用前面计算过的结果

```
int maxSubArray(int[] nums) {  
    if (nums == null || nums.length == 0) return 0;  
    int max = Integer.MIN_VALUE;  
    for (int begin = 0; begin < nums.length; begin++) {  
        int sum = 0;  
        for (int end = begin; end < nums.length; end++) {  
            sum += nums[end];  
            max = Math.max(max, sum);  
        }  
    }  
    return max;  
}
```

■ 空间复杂度: $O(1)$, 时间复杂度: $O(n^2)$

解法2 - 分治

■ 将序列均匀地分割成 2 个子序列

□ $[begin, end) = [begin, mid) + [mid, end)$, $mid = (begin + end) >> 1$

■ 假设 $[begin, end)$ 的最大连续子序列和是 $S[i, j)$ ，那么它有 3 种可能

□ $[i, j)$ 存在于 $[begin, mid)$ 中，同时 $S[i, j)$ 也是 $[begin, mid)$ 的最大连续子序列和

□ $[i, j)$ 存在于 $[mid, end)$ 中，同时 $S[i, j)$ 也是 $[mid, end)$ 的最大连续子序列和

□ $[i, j)$ 一部分存在于 $[begin, mid)$ 中，另一部分存在于 $[mid, end)$ 中

✓ $[i, j) = [i, mid) + [mid, j)$

✓ $S[i, mid) = \max \{ S[k, mid) \}, begin \leq k < mid$

✓ $S[mid, j) = \max \{ S[mid, k) \}, mid < k \leq end$



解法2 - 分治

```
int maxSubArray(int[] nums) {  
    if (nums == null || nums.length == 0) return 0;  
    return maxSubArray(nums, 0, nums.length);  
}  
int maxSubArray(int[] nums, int begin, int end) {  
    if (end - begin < 2) return nums[begin];  
    int mid = (begin + end) >> 1;  
    int leftMax = nums[mid - 1];  
    int leftSum = leftMax;  
    for (int i = mid - 2; i >= begin; i--) {  
        leftSum += nums[i];  
        leftMax = Math.max(leftMax, leftSum);  
    }  
    int rightMax = nums[mid];  
    int rightSum = rightMax;  
    for (int i = mid + 1; i < end; i++) {  
        rightSum += nums[i];  
        rightMax = Math.max(rightMax, rightSum);  
    }  
    return Math.max(leftMax + rightMax,  
        Math.max(maxSubArray(nums, begin, mid),  
            maxSubArray(nums, mid, end)));  
}
```

■ 空间复杂度: $O(\log n)$

■ 时间复杂度: $O(n \log n)$

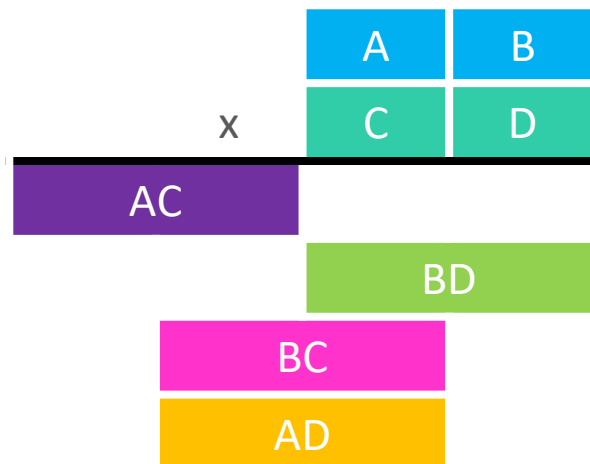
□ 跟归并排序、快速排序一样

□ $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

练习2 - 大数乘法

- 2个超大的数（比如2个100位的数），如何进行乘法？
- 按照小学时学习的乘法运算，在进行 n 位数之间的相乘时，需要大约进行 n^2 次个位数的相乘
- 比如计算 36×54

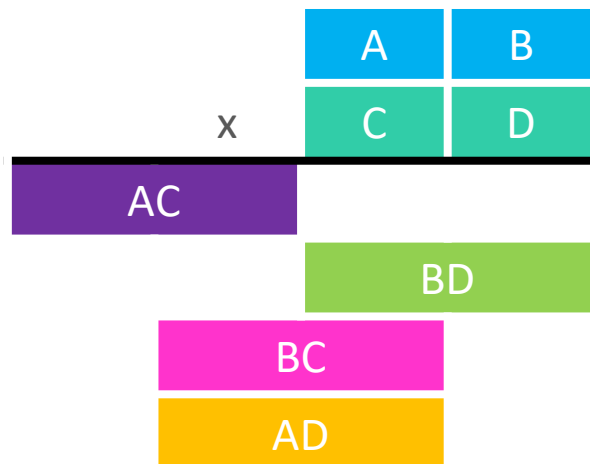
$$\begin{array}{r}
 36 \\
 \times 54 \\
 \hline
 24 \\
 120 \\
 \hline
 1944
 \end{array}$$



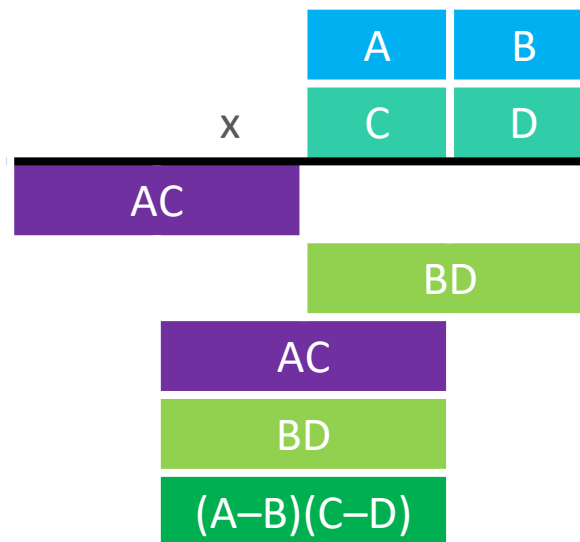
$$\blacksquare T(n) = 4T\left(\frac{n}{2}\right) + O(n) = O(n^2)$$

练习2 – 大数乘法

■ 1960 年 Anatolii Alexeevitch Karatsuba 提出了 Karatsuba 算法，提高了大数乘法的效率



■ $BC + AD = AC + BD - (A - B)(C - D)$



■ $T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O(n^{1.585})$