

Course 1 Week 1

第一门课主要是介绍分治算法的，这一周主要介绍了一些算法的基本概念，例如算法复杂度等等，下面分别回顾下。

Part 1:整数乘法以及Merge Sort

整数乘法

- Input: 两个数字n位的数字x,y
- Output: $x*y$

如果直接计算不必多说，中学里已经学过，这里老师介绍了另一种算法

Karatsuba Multiplication

记 $x = 10^{\frac{n}{2}}a + b, y = 10^{\frac{n}{2}}c + d$, 那么

$$xy = 10^n ac + 10^{\frac{n}{2}}(ad + bc) + bd$$

这要来看，我们只要计算 ac, ad, bc, bd 即可，考虑如下关系

$$\begin{aligned}(a+b)(c+d) &= ac + (ad + bc) + bd \\ ad + bc &= (a+b)(c+d) - ac - bd\end{aligned}$$

这说明不必分别计算 ad, bc ，而是可以把这两项的和当作整体，通过计算 $(a+b)(c+d), ac, bd$ 来计算出 $ad + bc$ ，

所以就有了如下算法

- $xy = 10^n ac + 10^{\frac{n}{2}}(ad + bc) + bd$
- 递归计算 ac
- 递归计算 bd
- 递归计算 $(a+b)(c+d)$

这个算法是一个典型的分治算法，下面再介绍一个经典的分治算法——Merge Sort

Merge Sort

算法介绍

Merge Sort 是用来排序的，之所以这里介绍它，是因为这个算法采取了分治的思路，先给出算法的输入输出

- Input: n个数字的未排序数组
- Output: 数组按递增顺序排列

再给出算法的伪代码

- 递归地对数组的前一半进行排序

- 递归地对数组的后一半进行排序
- 将两个数组归并(merge)为一组

归并的伪代码如下

Pseudocode for Merge:

<pre> C = output [length = n] A = 1st sorted array [n/2] B = 2nd sorted array [n/2] i = 1 j = 1 </pre>	<pre> for k = 1 to n if A(i) < B(j) C(k) = A(i) i++ else [B(j) < A(i)] C(k) = B(j) j++ end </pre> <p style="text-align: right;">(ignores end cases)</p>
--	---

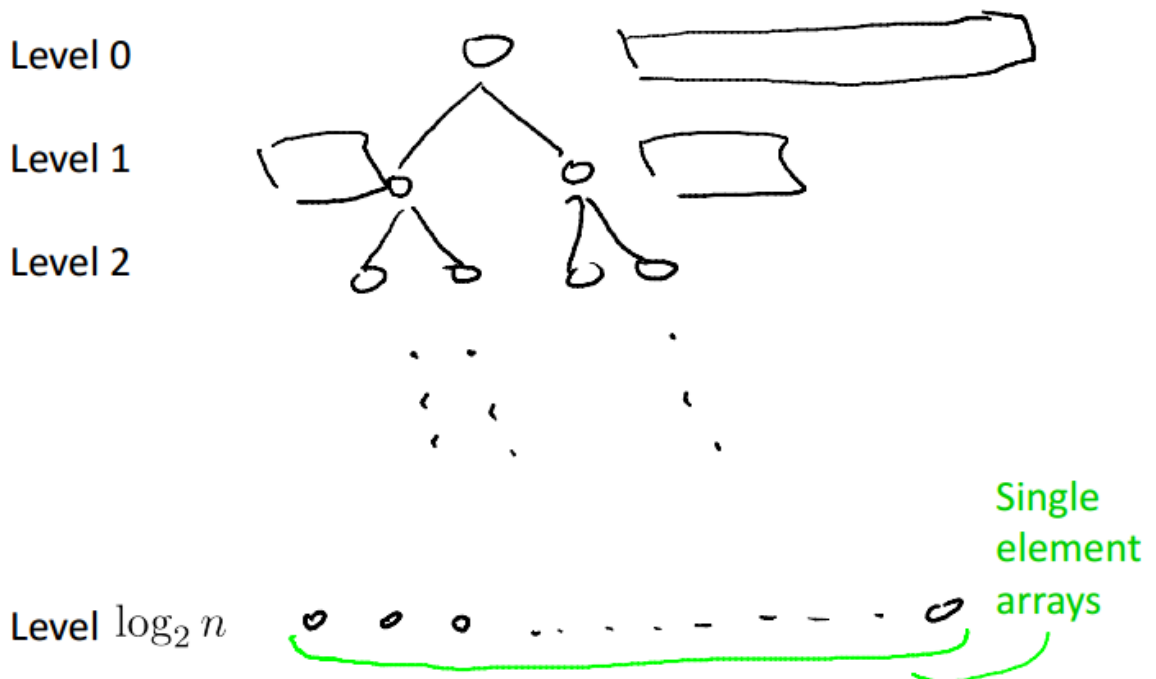
算法分析

有了算法，最重要的是分析它的运行效率，我们来讨论Merge Sort的时间复杂度。

先考虑每次merge的时间复杂度，假设merge的数组的长度为 m ，由上图可知， i, j 初始化需要2次操作，后面每次循环分别需要4次，分别为给 k 赋值，比较 $A(i), B(j)$ ，给 $C(k)$ 赋值， i, j 的自增运算，所以每次merge需要 $4m + 2 \leq 6m$ 次操作。

接着考虑递归的过程，方便起见，假设原数组的长度 n 为2的幂。由伪代码可知，每次将数组均匀拆分为两个长度为原有数组长度一半的数组，所以这个过程最多进行 $\log_2 n$ 次，整个过程可以由如下树状图所示

Proof of claim (assuming $n = \text{power of } 2$):



考虑第 j 层，有 2^j 个节点，每个节点有 $\frac{n}{2^j}$ 个数字，即第 j 层有 2^j 个子问题，每个子问题的大小规模为 $\frac{n}{2^j}$ ，所以第 j 层 merge 所需的时间复杂度为

$$2^j \times 6 \frac{n}{2^j} = 6n$$

因为一共有 $\log_2 n + 1$ 层，所以总共的时间复杂度为

$$6n(\log_2 n + 1)$$

最后，为了叙述方便，这里再介绍算法分析常用的一些符号。

常用记号

Big-Oh Notation

定义：

$$T(n) = O(f(n)) \text{ 等价于存在 } c, n_0 > 0, \text{ 使得}$$
$$\text{对任意的 } n \geq n_0, T(n) \leq cf(n)$$

这里比较关键的一点是 c, n_0 与 n 无关。

Omega Notation

定义：

$T(n) = \Omega(f(n))$ 等价于存在 $c, n_0 > 0$, 使得
对任意的 $n \geq n_0, T(n) \geq cf(n)$

这里也是 c, n_0 与 n 无关。

Theta Notation

定义:

$T(n) = \theta(f(n))$ 等价于 $T(n) = O(f(n))$ 和 $T(n) = \Omega(f(n))$

这也等价于

存在常数 c_1, c_2, n_0
任取 $n \geq n_0, c_1 f(n) \leq T(n) \leq c_2 f(n)$

例子

这里举一个比较重要的例子

$\max\{f(n), g(n)\} = \theta(f(n) + g(n)), f(n) > 0, g(n) > 0$

从两个方向证明

$\max\{f(n), g(n)\} \leq f(n) + g(n)$
 $2 \times \max\{f(n), g(n)\} \geq f(n) + g(n)$

所以

$\frac{1}{2}(f(n) + g(n)) \leq \max\{f(n), g(n)\} \leq f(n) + g(n)$
 $\max\{f(n), g(n)\} = \theta(f(n) + g(n))$

Part 2:习题

选择题

选择题1

3-way-Merge Sort: 假设在Merge Sort的每一步中不是分成两半, 而是分成三分之一, 对每部分进行排序, 最后使用三向合并子程序将它们组合起来。这个算法的整体渐近运行时间是多少? (提示: 请注意, 合并步骤仍然可以在 $O(n)$ 时间。)

- n
- $n^2 \log(n)$
- $n(\log(n))^2$
- $n \log(n)$

这里和课件中的区别为树状图为三叉树, 即第 j 层有 3^j 个子问题, 每个子问题的大小规模为 $\frac{n}{3^j}$, 由于长度为 m 的数组每次Merge操作需要的时间复杂度为 $O(m)$, 所以第 j 层的时间复杂度为

$$O(3^j \times \frac{n}{3^j}) = O(n)$$

因为为三叉数，所以一共有 $O(\log n)$ 层，总共的时间复杂度为

$$O(n \log n)$$

选择题2

给你两个函数 f, g 满足 $f(n) = O(g(n))$ ，那么 $f(n) * \log_2(f(n)^c) = O(g(n) * \log_2(g(n)))$ 是否成立，其中 c 为正常数，你可以假设 f, g 非递减并且都大于1

- 有时是，有时不是，取决于 f 和 g
- 错误
- 有时是，有时不是，取决于常数 c
- 正确

因为 $f(n) = O(g(n))$ ，所以存在 n_0, k ，使得当 $n \geq n_0$ 时，

$$f(n) \leq kg(n)$$

因为 f, g 都大于1，当 $n \geq n_0$ 时

$$f(n) * \log_2(f(n)^c) = cf(n) * \log_2(f(n)) \leq ckg(n) * \log_2(kg(n)) = ckg(n) * (\log_2 k + \log_2 g(n)) \leq m * g(n) * \log_2 g(n)$$

上述不等式关系可能不太严格，但是这里主要估计数值级，无伤大雅，所以这个结论是正确的。

选择题3

两个非递减的正函数 f, g 满足 $f = O(g(n))$ ，是否满足 $2^{f(n)} = O(2^{g(n)})$

- 正确，如果 $f(n) \leq g(n)$ 对于所有足够大的 n
- 有时正确
- 从不正确
- 总是正确

先举一个例子： $f(n) = n, g(n) = 2n$ ，那么 $f = O(g(n))$ ，但是

$$\begin{aligned} 2^{f(n)} &= 2^n \\ 2^{g(n)} &= 2^{2n} = 4^n \end{aligned}$$

所以 $2^{f(n)} = O(2^{g(n)})$ 成立

再举一个例子 $f(n) = n, g(n) = 2n$ ，那么 $f = O(g(n))$ ，并且

$$\begin{aligned} 2^{f(n)} &= 2^{2n} = 4^n \\ 2^{g(n)} &= 2^n \end{aligned}$$

所以 $2^{f(n)} = O(2^{g(n)})$ 不成立

所以选项2是对的，此外从上述两个例子可以看出，该结论成立与否与 f, g 的大小有关，现在假设 $f(n) \leq g(n)$ 对于所有足够大的 n ，那么

$$2^{f(n)} \leq 2^{g(n)}$$

从而 $2^{f(n)} = O(2^{g(n)})$

因此这题前两个选项均正确。

选择题4

k-way-Merge Sort。假设你得到了 k 个排好序的数组，每个都有 n 元素，并且你希望将它们组合成一个有 kn 个元素数组。考虑以下方法。使用讲座中讲授的merge方法，merge前2个数组，然后用merge后的前两个数组merge第3个数组，依此类推，直到你把第 k 个数组merge完。这个连续合并算法的运行时间是多少？作为 k, n 的函数（可选：您能想到更快的方式来进行k-way合并过程吗？）

- $\theta(n^2 k)$
- $\theta(nk)$
- $\theta(n \log(k))$
- $\theta(nk^2)$

merge两个数组的时间和两个数组中较长的长度成正比，所以上述算法中，第 i 次merge的时间复杂度为

$$\theta(in)$$

总共的时间复杂度为

$$\sum_{i=1}^k \theta(in) = \theta(k^2 n)$$

更快的算法应该是将 k 个数组一起merge，这样的时间复杂度为 $\theta(kn)$

选择题5

按递增的顺序排列以下函数， $g(n)$ 在 $f(n)$ 之后当且仅当 $f(n) = O(g(n))$

- a) 2^{2^n}
- b) 2^{n^2}
- c) $n^2 \log(n)$
- d) n
- e) n^{2^n}

由数学知识可知，顺序为

$$eabcd$$