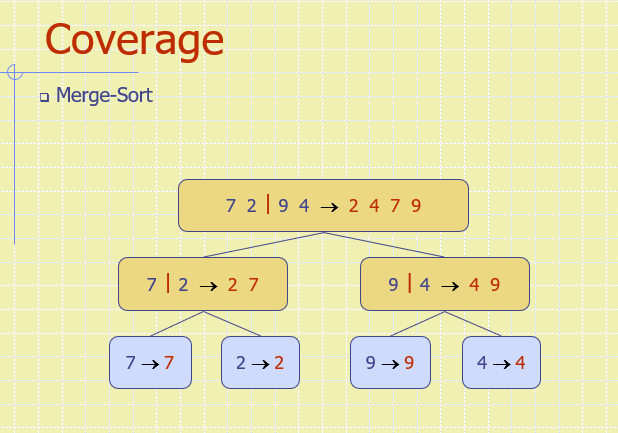
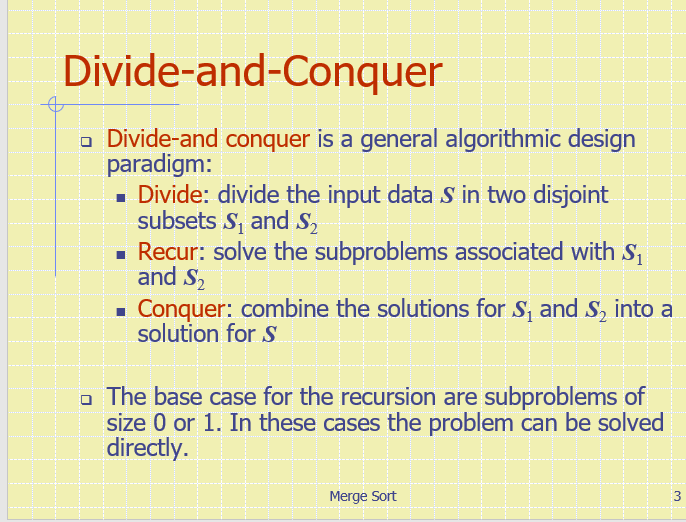
Merge Sort



具体逻辑就是 拆开，合并



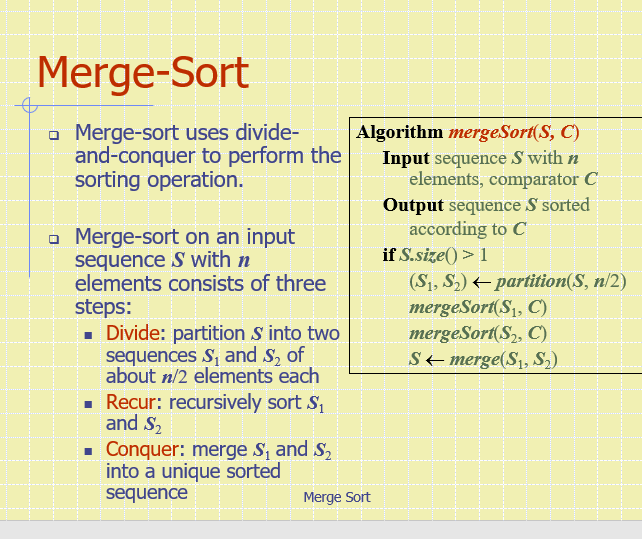
divide and conquer 分而治之.... 是一个常用算法思想

divide:把输入数据S 分成两个分离的子集 S1,S2

Recur:把对应的问题分成子问题，S1S2分别解决

Conquer:把S1S2的解合并，也就是S1的解

Basecase:是size0或1的子集，



merge-sort 使用divide-and-conquer法来进行sort操作

merge-sort 有三步：

devide:把S等分成S1,S2

Recur:对S1 S2进行SORT（这里会使用recursion）

Conquer:把S1 ,S2合二为一

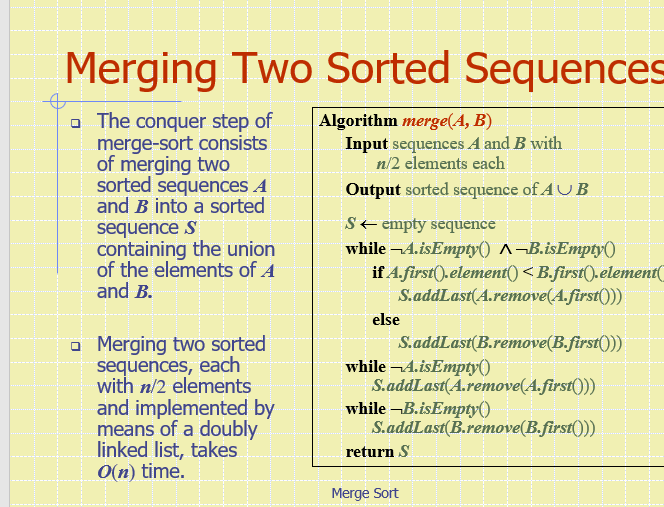
算法翻译：

当S 的 size大于1， 把他Partition 成S1,S2

对s1s2分别使用MergeSort

把S1 S2 MERGE成S

把两个Sorted Sequence Merge到一起的方法



当A不等于空且B不等于空时

如果A的firstelement小于B的firstelement

那么就把A的第一个remove并且加到S的最后

不然就把B的第一个remove并且加到S的最后

如果A空了

把A的第一个REMOVE并加到S的最后

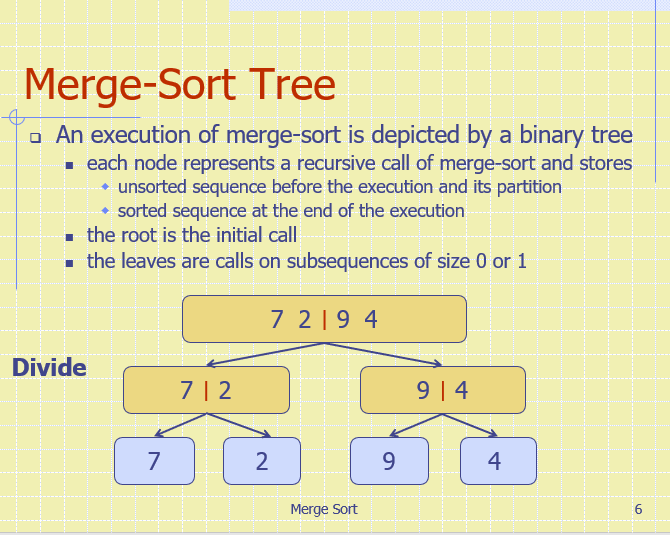
如果B空了

把B的第一个REMOVE并加到S的最后

注意A与B本身是SORTED的

总共用时O（n）

Merge Sort tree



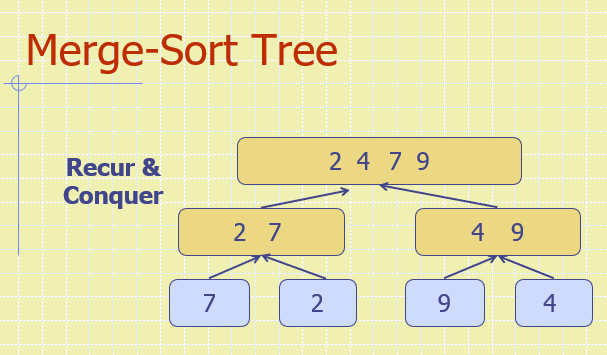
merge-sort的执行可以用binary tree表示

每一个node代表一个merge-sort的recursive call （就是对分出来的子array再进行merge-sort）， 比如说7|2,就代表着对72进行merge sort

然后每个node还存储着： 执行与分离之前的原始unsorted sequence  
 在执行Merge-sort以后的sorted sequence

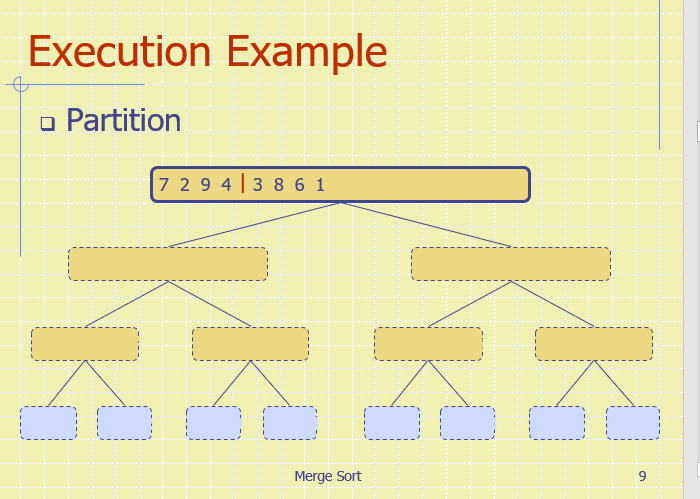
root是第一次call merge-sort

leaves 是size为0到1的子sequence 对merge-sort的call

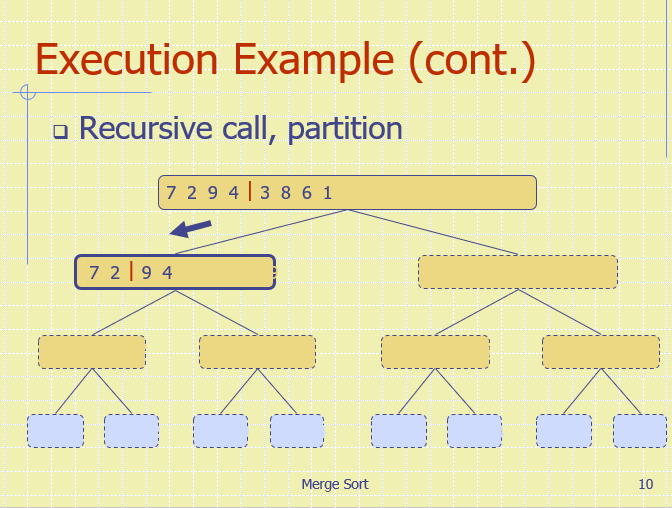


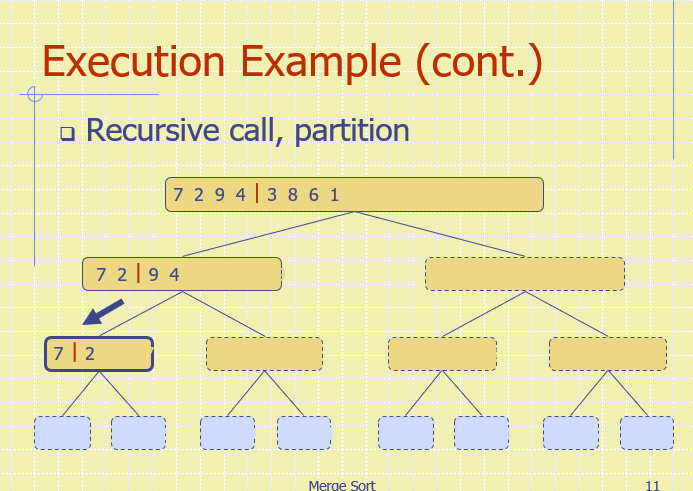
recur&conquer以后的景象

partition:分割

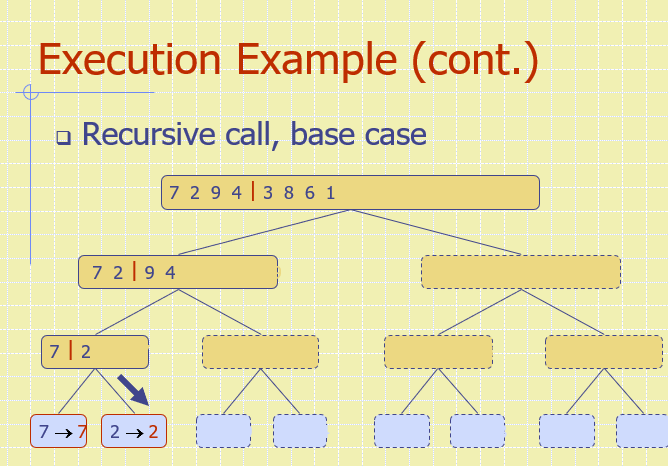


recursion+分割

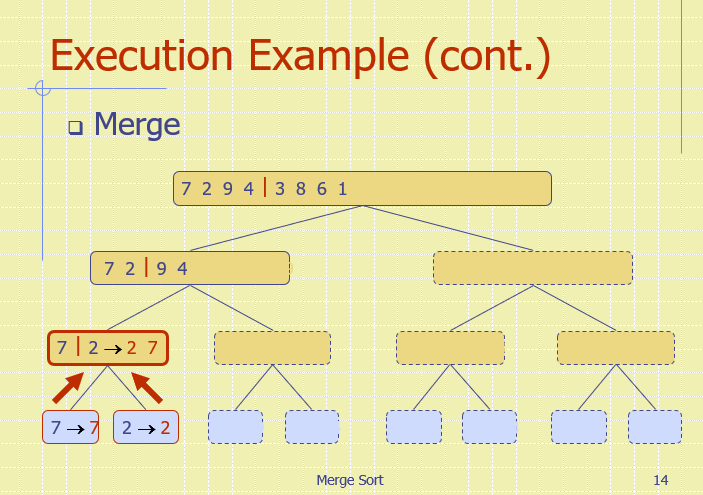




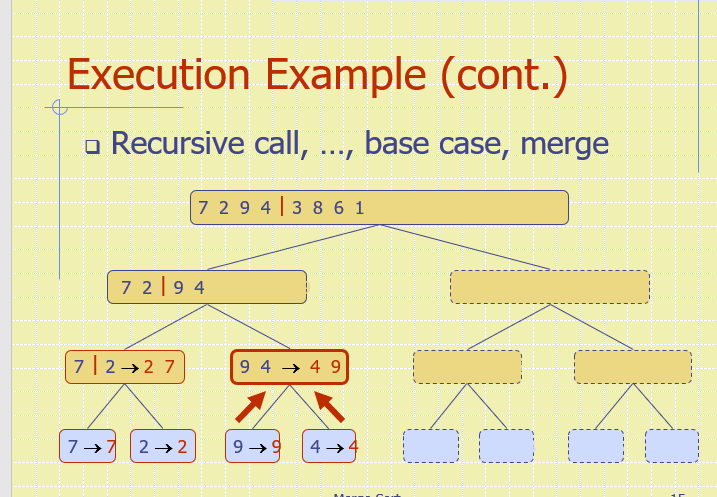
recursive call+ base

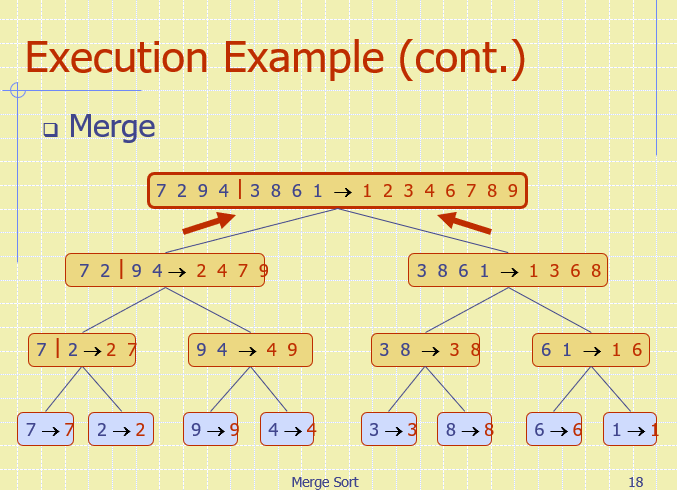


merge

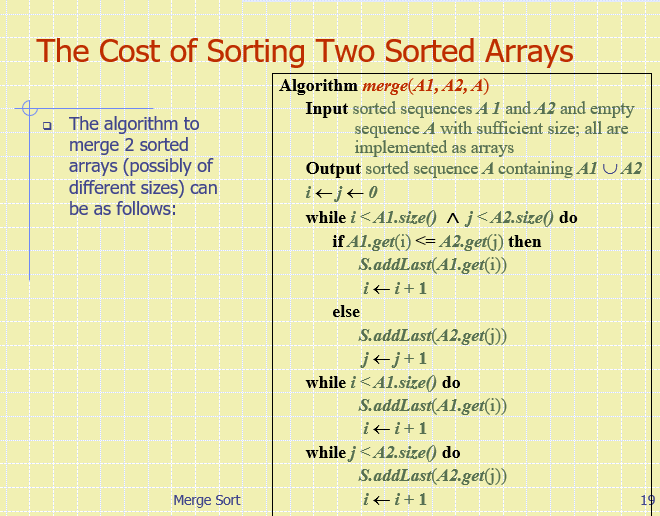


同理





把两个sorted array merge起来的算法



A是有着充足空间的空sequence

当i小于A1.SIZE 且j小于A2.size的时候（两个sorted array仍然没有被取完）

A1大A1 ADD LAST, i++

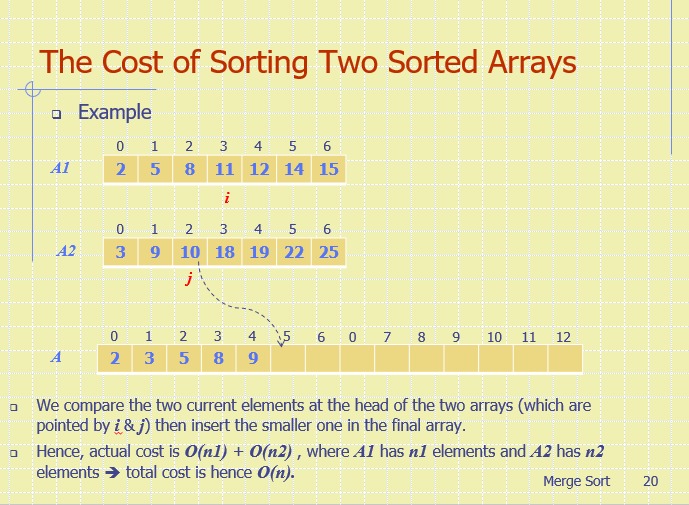
A2 大 A2 ADD LAST ,j++

如果i< A1.SIZE(A2被取完了)

a1.addlast,i++

如果j<A2.size （A1被读取完了）

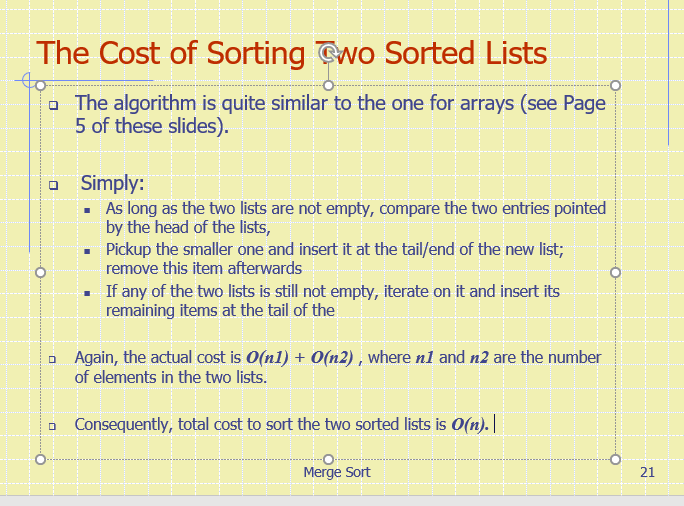
a2.addlast,j++

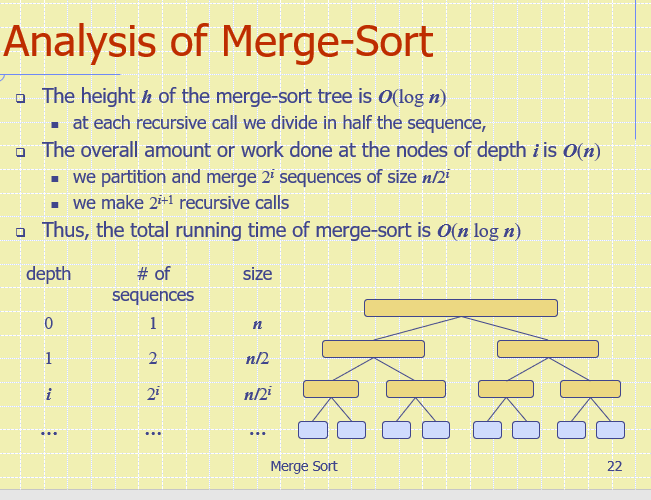


所需要的On， 我们比较两个array当前head并且把较小的那个插入final array，所以总cost就是On1+On2，n1n2是A1A2有的element，总cost是On

Sorted list

无论是算法，还是cost，都与array一样





总分析： 一个含有size为n的sequence tree的高度h是Ologn

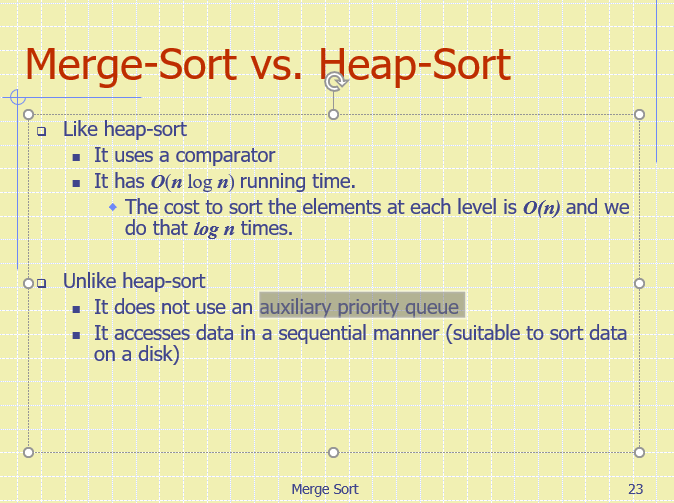
因为每次都要除以2，

每层所有node加起来总共要做的work是On

假设是第i层，这时已经分了2^i组，每组有n/2^i个元素，我们就要进行2^i+1次call，

2^i+1 \*n/2^i =2n=n

+  
因此总消耗是nlogn



与heap sort的不同，虽然都是ONLOGN并且使用comparator，但是他不需要使用auxiliary PQ

它以顺序的方式访问数据(适合于对磁盘上的数据进行排序)

