Sorting lower bound

在之前一课我们提到了，comparison-based sorting至少也有nlogn的时间来sort 一个 n element的sequence

那么我们能超过nlogn吗

答案是可以，但是对Input sequence 有特殊要求‘’

即便如此，这样的场景在实践中也经常存在，因此研究这样的算法是值得的。

Bucket Sort

假设S 是一个有着n个entry的sequence，并且Key都被限制在[0,N-1]，这个N大于1

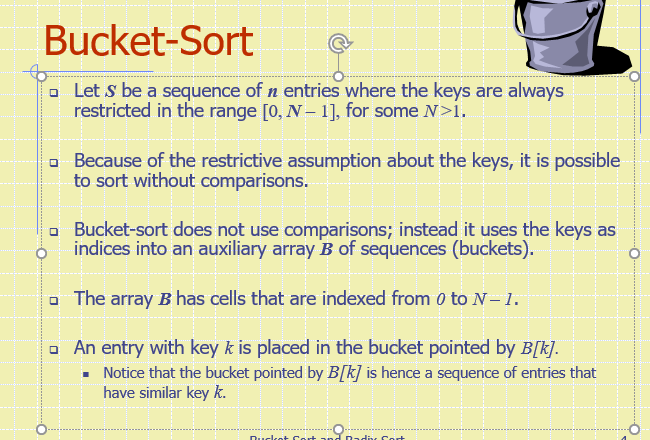
由于我们对于key本身是有限制的，所以不用比较就可以排序

Bucket-sort不使用比较;相反，它使用key作为array B的辅助sequence(bucket)的索引。

array B有从0到N-1的index个cell

一个有着key为k的entry被放入B[k]所指的bucket内

注意：B[k]所指的bucket因此是一sequence的有着同样Key的entries



第一步，把sequence清空，通过把每个entry(k,v)移到对应的bucketB[k]

第二步，把i=0,...N-1 把每个B[i]里的entry 移到sequence S的尾端

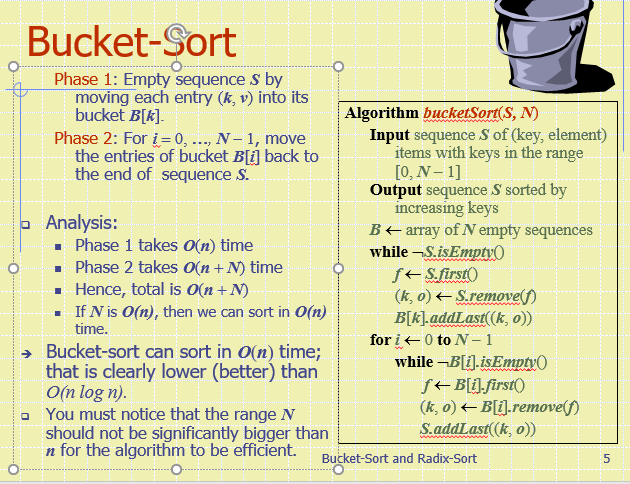
分析：第一步需要O（n） time

第二步需要O（n+N） time

因此，总共时间是O（n+N）

然而N又是O（n），那么sort in O(n) time

必须要注意的是，为了让算法有效，N不能显著大于n

.算法翻译，一个有着key的sequence S, Key的范围是0到N-1

输出：一个sorted sequence S, key升序

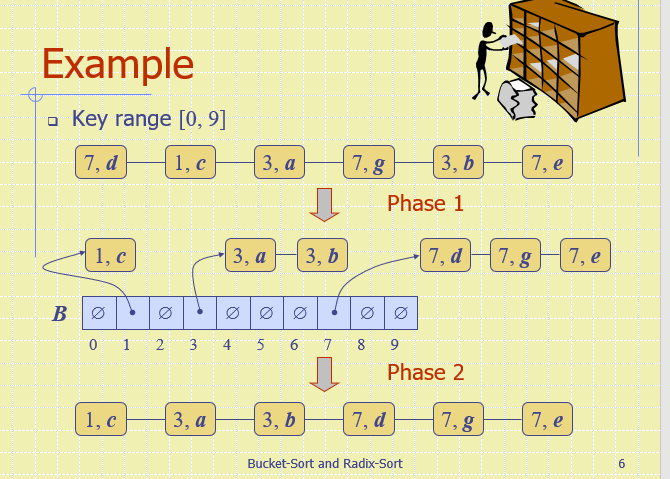
B，一个长度为N的空sequence

当S不是empty的时候，把S的第一个指定成f，然后remove f并存在（ko）这个entry里，B[k].addLast

当从i到0到N-1,

当Bi不是empty，取出来第一个，然后加到S里

循环N次



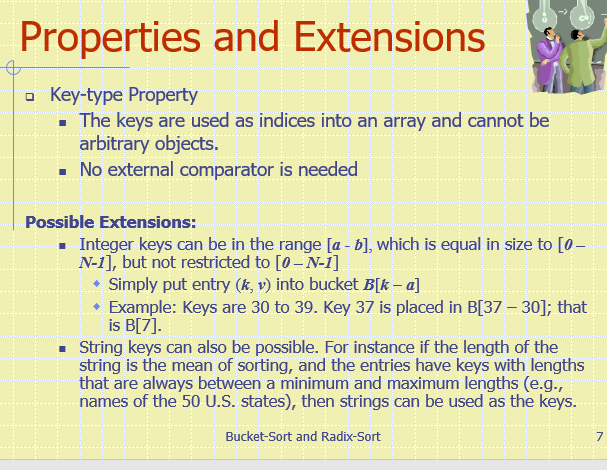
第一步，拆，第二步装，使用辅助sequence B

Properties and extensions

key-type property：key的性质

key是用来作为对辅助array的索引，所以不能是任意Object

不能使用external comparator



Possible extensions， 整数key可以是[a,b]之间，size=【0,N-1】,但并不固定在这个范围内

换句话说，整数key的范围的长度必须等于0到N-1的长度，但是key的范围可以再0到N-1之外

我们只需要把entry(k,v)存储在B【k-a】就成

比如说key从30到39，37就被存储在37-30=7 ，存储在B[7]就完事儿

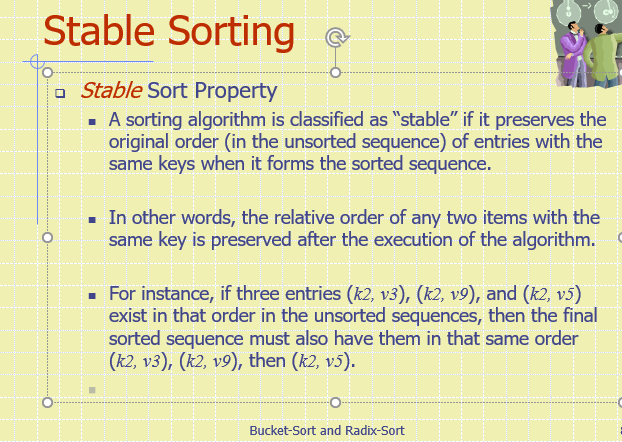
String类型的key其实也可以存储，只要我们把string的长度看作是整数key的大小

Stable Sorting

一个排序算法被定义为稳定"stable"当：在它形成sorted sequence的过程中，如果两个entry有着同样的key，那么在新sorted sequence中，他们的前后顺序是与unsorted sequencer相同的

换句话说，任意两个相同key的item的相对顺序在算法中被保持

例子，k2,v3,k2,v9,k2,v5，在final中，一定排序还是 k2,v3,k2,v9,k2,v5



我们之前对bucket-sort的非正式描述其实不能保证stability稳定性

1.我们没有限制entry是怎样从unsorted sequence提取出来的

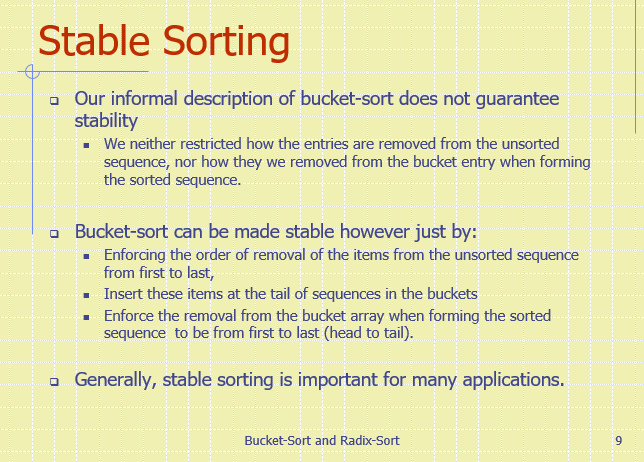
2.我们没有限制entry是怎样从辅助的bucket array中提取回去的

通过以下方法，我们可以让他stable:

1.强制执行在unsorted sequence中removal item的顺序：从前到后

2.每一次insert:，先找到自己所属的bucketB[k]，然后插入B【k】这个list的tail

3.强制执行removal回去，从前到后，从head到tail



Lexicographic Order 词典顺序

d-tuple元组:一个由d个key组成的sequence，ki：这个tuple的 i-th dimension次元

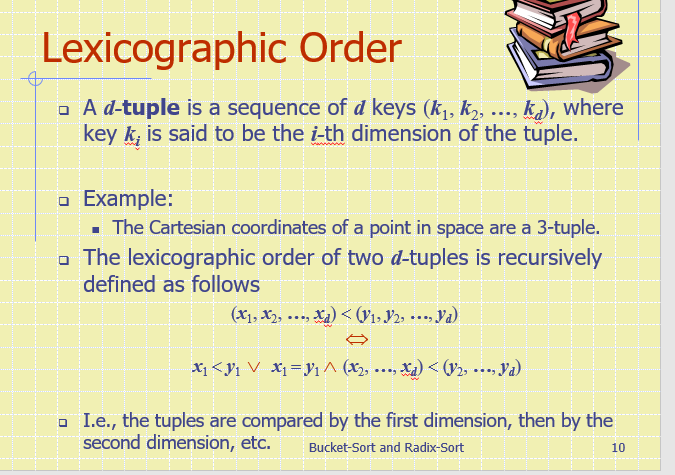
比如一

个笛卡尔坐标就是空间里的一个点，他是3-tuple的 （三维的）

两个d tuple之间的lexicographic order 的递归定义如下，

x1<y1或x1=y1^(x2,....xd)<(y2,...yd)

换句话来说，比较lexicographic order，先比较第一个，如果相等，比较第二个，如果再相等比较第三个...直到比较到最后

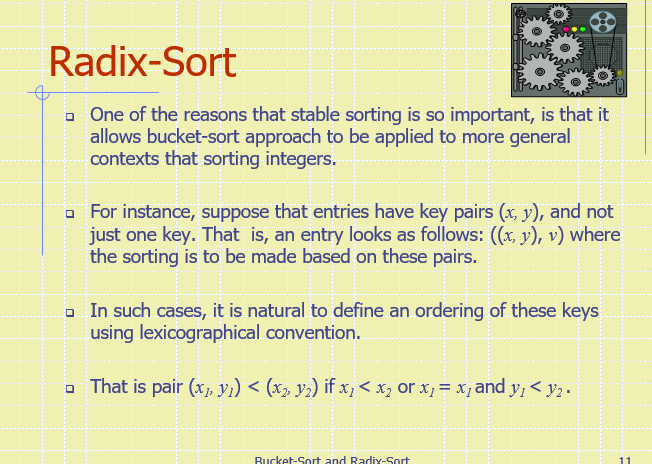


Radix-sort

stable sort如此重要的原因之一是，它允许bucket sort被应用到更普遍的需要对整数排序的场景中

例如，假设一个entry的key 不只是一个单独integer key，而是这个形式，那么我们比较key的顺序的时候用的就是lexicoraphical convention

如果

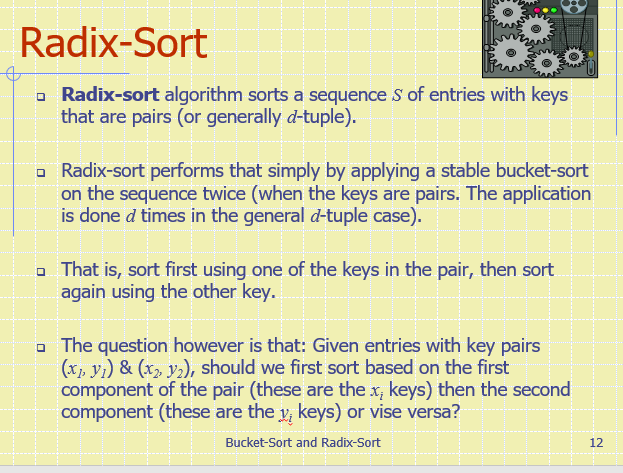


radix-sort算法对一个拥有 pair-key的 entry (或者更高的d-tuple) 的sequence S进行排序

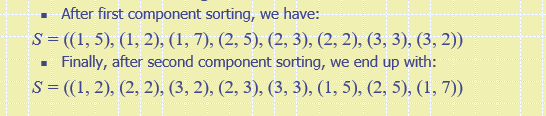
对于pair key，就直接使用两次stable bucket-sort

第一次取其中一个key来sort，第二次取另外一个key来sort

问题在于：当给定key pair，我们应该先sort x还是y

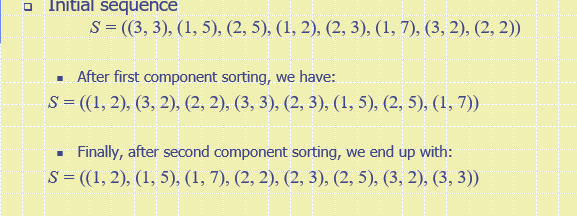


例子，先sort 第一个，再sort第二个



显然不是 lexicographically sorted

先sort第二个再sort第一个

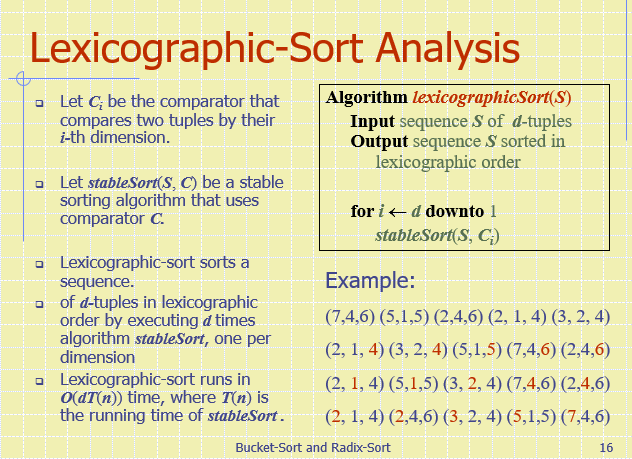
  
是lexicographically sort

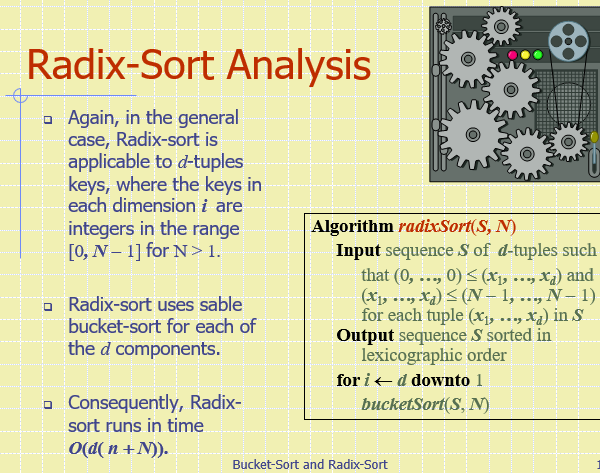
注意，第二次sort是取决于第一次sort以后的

以先二后1为例，如果是sort原始的那么就应该是



对于radix sort来说，我们要注意的最重要的一点就是，永远从最后面开始sort

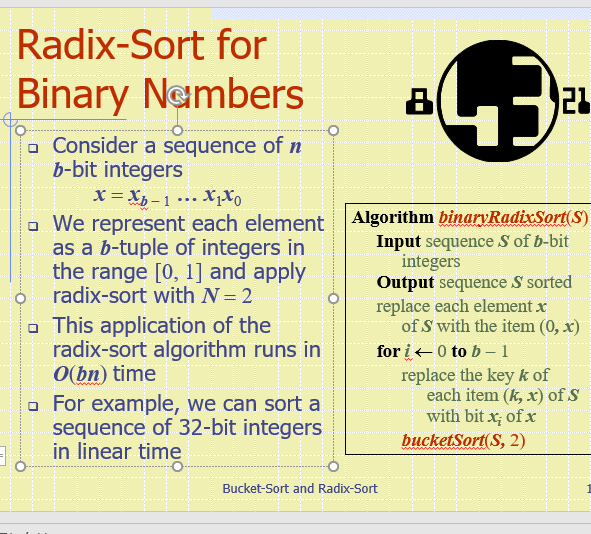




对于radix sort来说，每一个d tuple的范围要在[0,N-1]之间

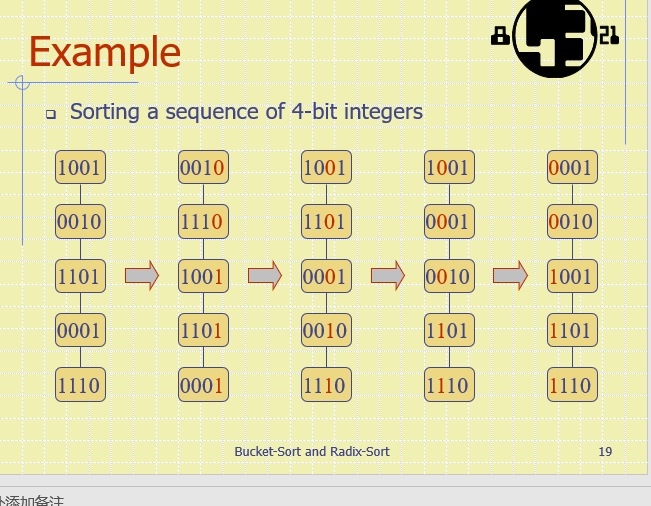
radix-sort 对每一个d component都用bucket sort，总用时为

对binary number二进制数使用radix sort



我们把每个element表示成b tuple，range是0到1，N=2

那么radix sort算法总共需要Obn时间 (线性时间)



先排最后一位，再排倒数第二位，....