

Quick-sort 是一个基于divide-and-conquer 的随机sorting algorithm

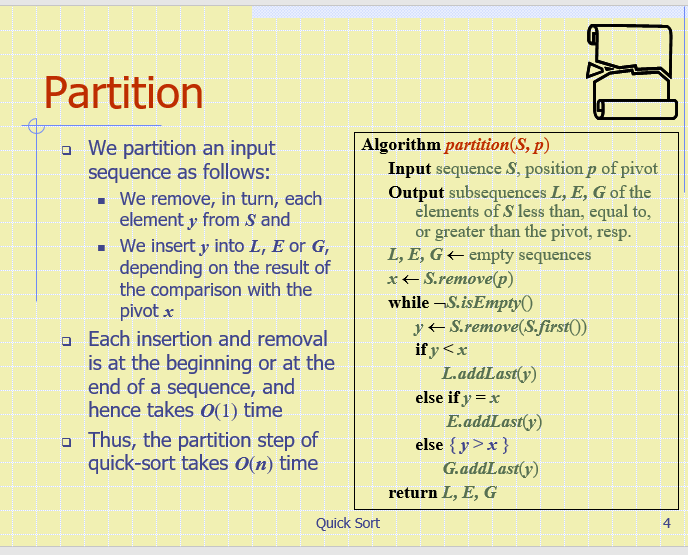
divide:选取随机的一个element x（叫做pivot）然后把S分割成

L :小于x的element  
E=X的element(包括x)

G 大于x的element

Recur:给L和G sort

Conquer:把LEG合并到一起



切割，

我们依次按顺序把每个element y放入S

然后我们把Y分别插入L ,E,G （取决于与pivot X的比较）

每一个insertion和removal 都用时O1

因此总共的partition step用时On time

算法翻译;

输入：sequence S, pivot的位置p

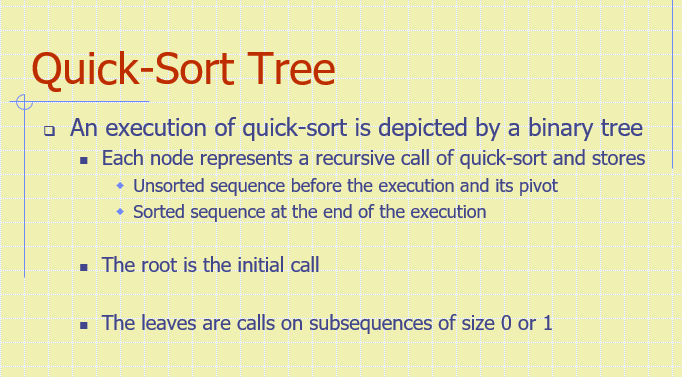
输出： LEG

设置empty sequence LEG

把x设置成S.remove (p),现在x指代Pivot

当S不是empty

remove first,然后分别addlast



quick-sort的执行由binary tree表示

每个node代表了一个quick-sort的recursive call

并且每个Node存了 1.还未排序的sequence与pivot

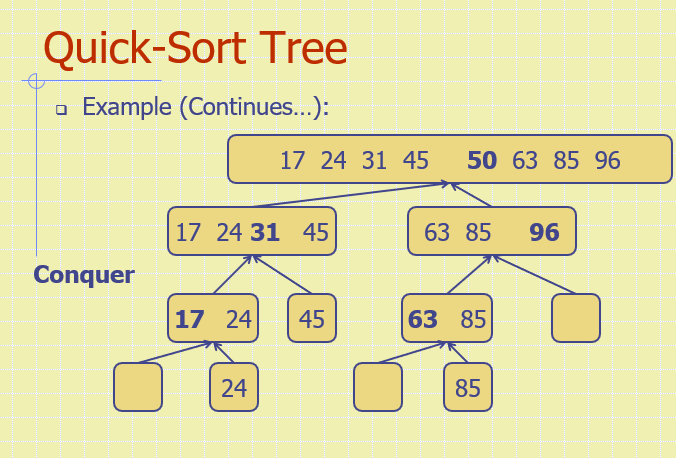
2.sorted sequence

root是quick-sort初始call

leaves 是size 是为0或1的subsequences的 quick-sort call

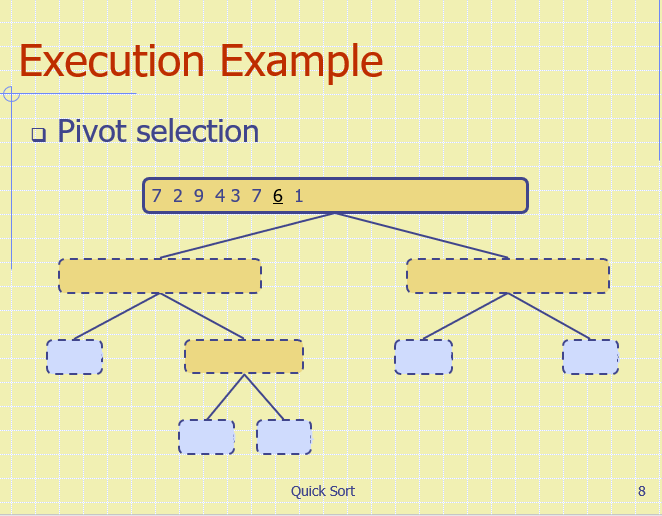
、

选取了一个pivot单独存到末尾， 然后小于他的放在左Node，大于他的放在右Node

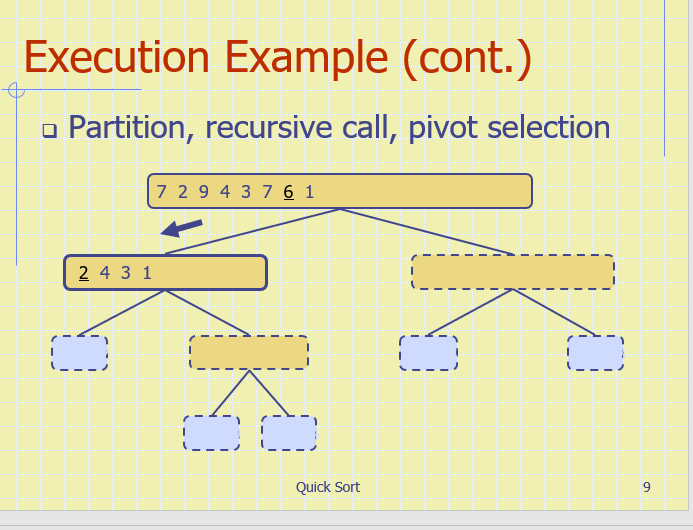


合并的时候就大于他的放在pivot的右边，小雨的就放在左边，注意是一组一组sorted sequence的插入

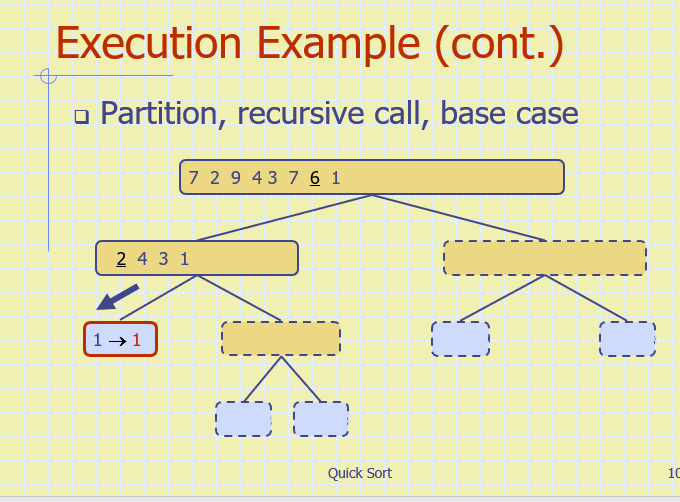
1.选取pivot

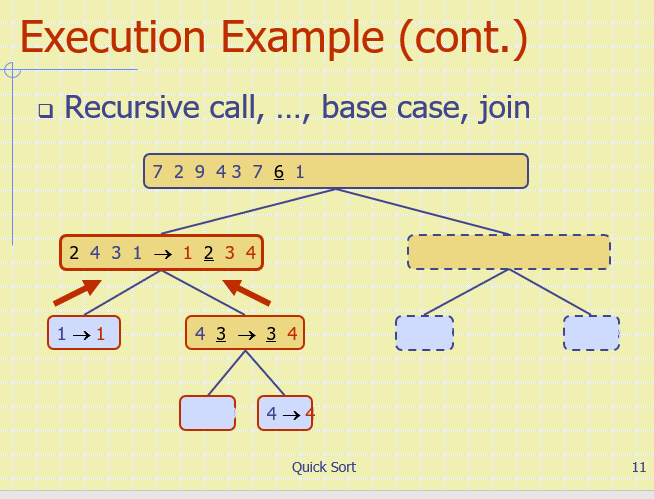


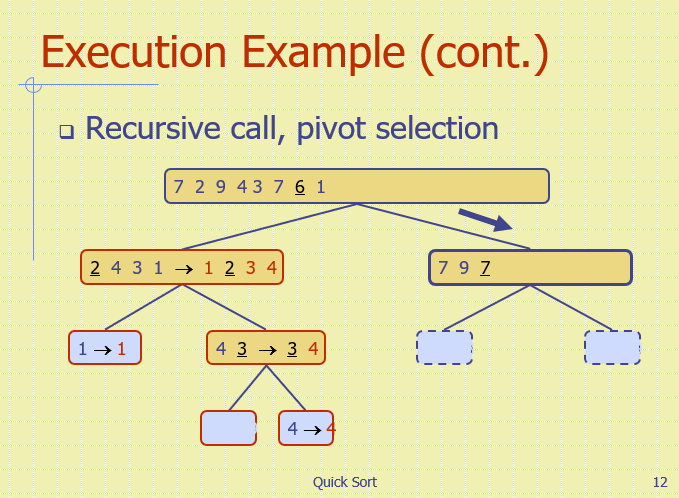
2.切割，pivot selection ，先做小于他的

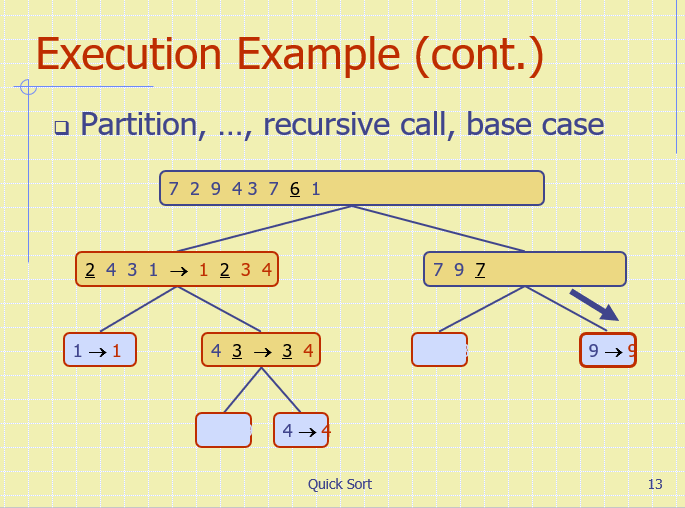


切割，小于他的已经到达base case,开始切割大于他的

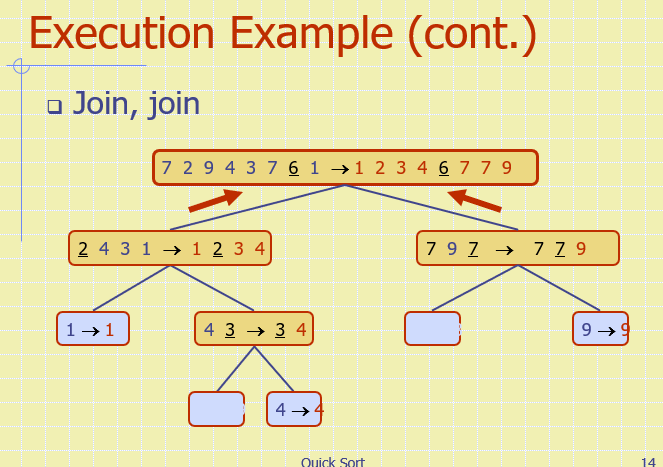








可以看见有相等的我们保留在原Node里



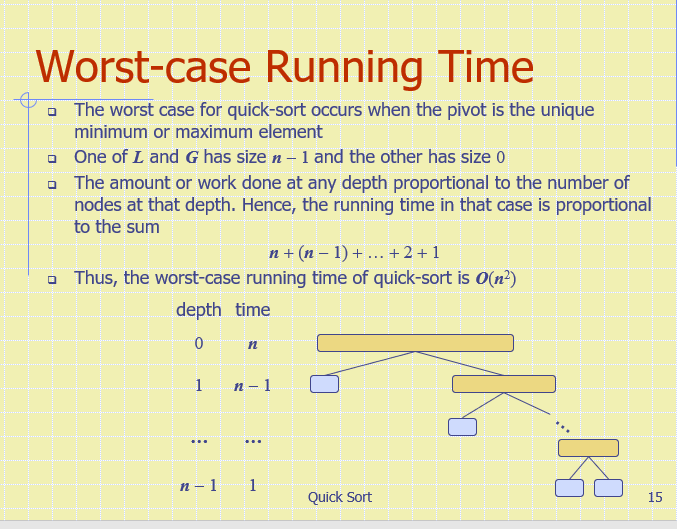
最差的情况：

pivot始终是最小的/最大的element

一个L或G的size是n-1,另一个是0

每一个需要做的时间就是他的深度

n+n-1+...1，复杂程度就是n^2了



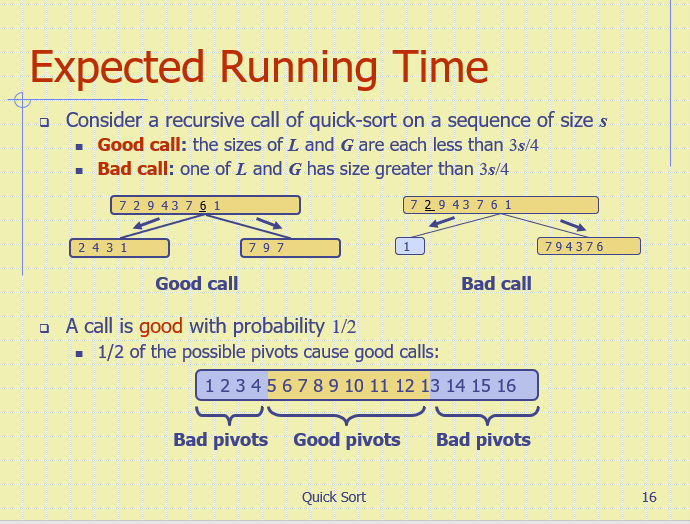
理论running time，

假设上一个node的size为s

good call:L和G的size都小于3S/4

bad call:一个 L或G的size大于3S/4

一个call good的几率是1/2



理论运行时间：

概率性事实:为了得到k次正面，抛硬币的期望次数是2k

那么对于一个depth为i的node，我们期望:  
i/2次ancestor 的call是good call

当前input sequence最大size是（3/4）^(i/2) \*n ,n是总list长度

那么：什么时候Input size能到1呢

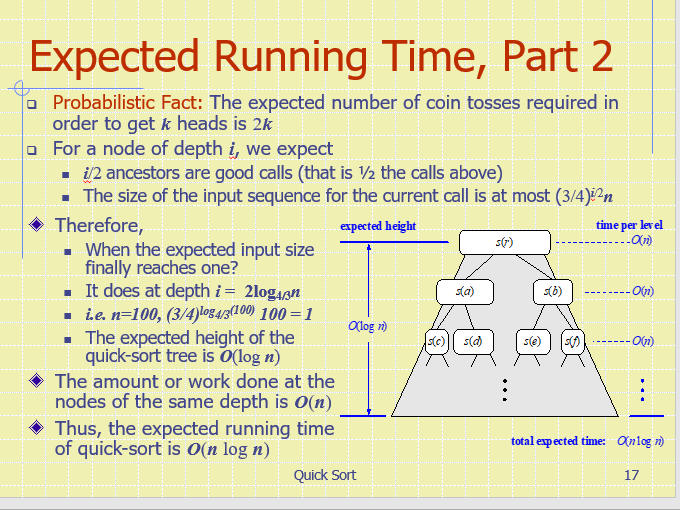
在I=2 log4/3n的时候

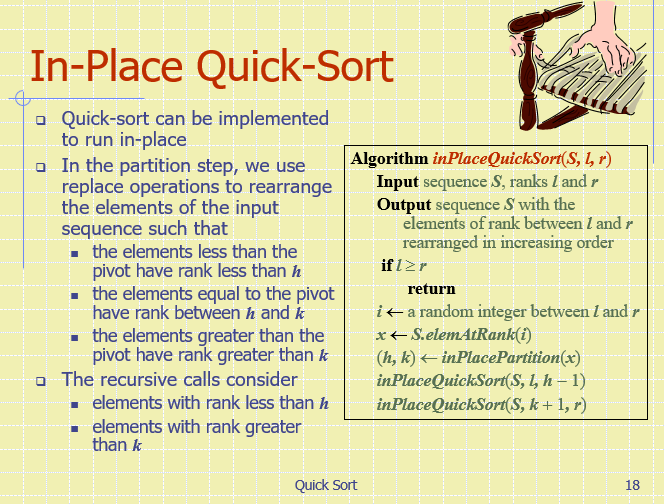
假设n=100, 3/4 ^log4/3 100 乘以100=1

因此我们期待的height是LOgn 因为到1的时候高度是2log 4/3 n，而求复杂度的时候不管下面

每一层要干的活是On

那么总共的running time 就是NLOGN





In place quick sort

quick sort可以由in-place implement

在分割的时候，我们使用replace操作 来重新分布element，

1.比pivot小的element 的rank小于h

等于pivot的rank在H与k之间

大于pivot的rank大于k

recursive calls：

rank小于h的element，大于k的element

算法翻译：

输入sequence S, index L和r

输出 sequence S, element的index在L与r之间并且升序

如果l>=r， return

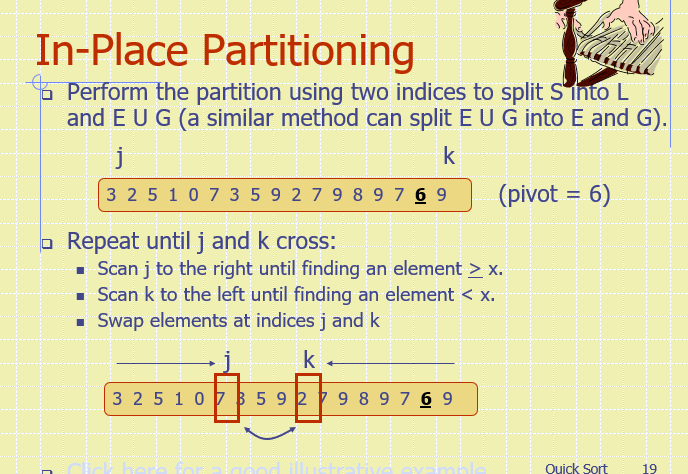
不然，i是l到r之间随便一个index

x就是index i上的元素也就是Pivot

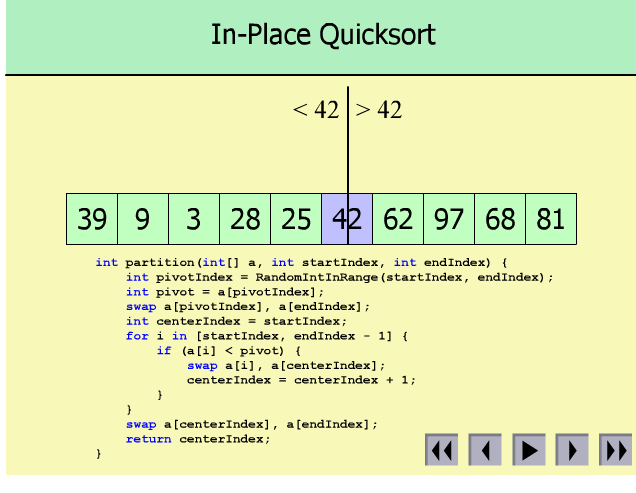
然后对等于x的元素设置H到k的Index

s,l,h-1在inplace quick sort

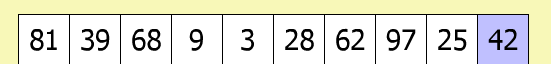
.....



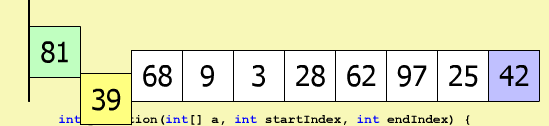
设置6位pivot，



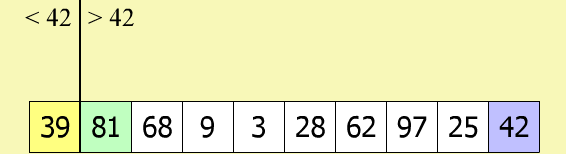
首先，选取任意一个数，把它放到最后

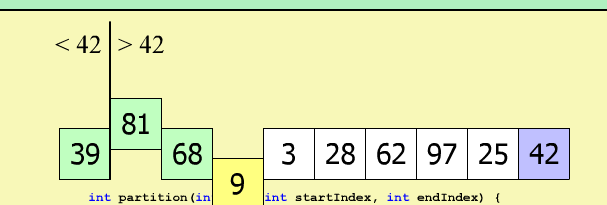


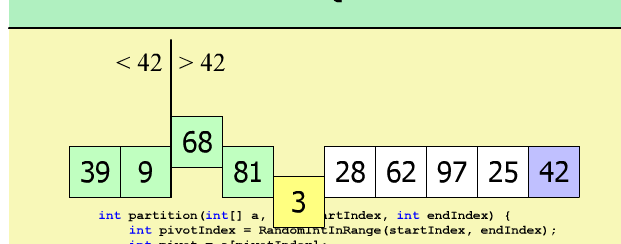
从一个大于Pivot的数开始数，如果遇到了一个小于Pivot的数，swap



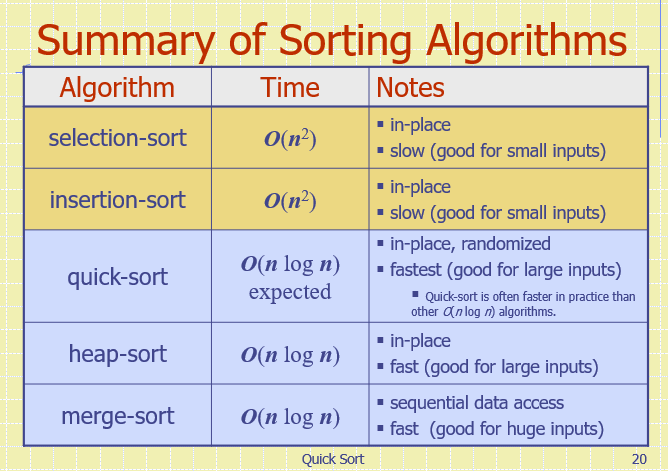
当你成功换一次，指示线 前进一格







最后停止的地方就是42要呆的地方



quick sort expected是nlogn最快，实际上最坏是n^2