选择排序

《数据结构》

指院网络工程教研中心 陈卫卫

	49	38	65	97	76	13	27	49‘
i=1	49	38	65	49‘	76	13	27	[97]
i=2	49	38	65	49‘	27	13	[76	97]
i=3	49	38	13	49‘	27	[65	76	97]
i=4	27	38	13	49‘	[49	65	76	97]
i=5	27	38	13	[49‘	49	65	76	97]

选择排序

基本原理：

从n个元素中选出一个最大（小）元素，把它调到序列末（前）端，再从剩下的n-1个元素中选出最大（小）元素…反复如此，直到只剩1个元素时，完成排序。

选择排序

如何选择最大（小）元素

- 简单选择排序
- 堆排序

学习目标和要求

1、准确复述选择排序的基本原理

2、编写简单选择排序算法

2、知道算法的时间复杂性和稳定性

[例]　有一组待排数据：49，97，76，13，27，49'，用**简单选择排序**方法排序。

49　97　76　13　27　49'

交换后：　49　49'　76　13　27　97

交换后：　49　49'　27　13　76　97

交换后：　13　49'　27　49　76　97

交换后：　13　27　49'　49　76　97

交换后：　13　27　49'　49　76　97

简单选择排序

简单选择排序程序段：

```
1.  for(k=n-1;k>0;k--)
    {   //k控制选择的遍数，是每遍选择的终点
2.       i=0;                //i记录最大元的下标
3.       for(j=1;j<=k;j++)
              if(a[j]>a[i]) i=j;    //选择最大元素a[i]
4.       t=a[i], a[i]=a[k], a[k]=t;    //将最大元交换到最后
    }
```

简单选择排序的时间复杂性

$$T(n)=\sum_{i=1}^{n-1} i=\frac{n(n-1)}{2}=O(n^2)$$

各种排序方法比较

排序方法	平均情况	最坏情况	辅助存储	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
希尔排序	与增量序列有关：$O(n^{3/2})$,$O(n(\log n)^2)$			×
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
快速排序	$O(n\log n)$	$O(n^2)$	$O(\log n)$	×
简单选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	×

问题分析

- 效率不高的主要原因是反复顺序比较选择，存在大量的冗余比较。
- 比如：

  1）元素x与y比较一次之后，就不必再比较它们。

  2）假如x与y，y与z都进行过比较，而且比较结果有：x>y而且y>z，那么x与z也用不着比较了，因为根据比较结果的“传递性”，必有x>z。

学习目标和要求

1、准确复述堆排序的基本原理

2、编写堆排序算法

3、分析堆排序算法的特点

4、知道算法的时间复杂性和稳定性

1. 堆排序（heap sort）

由J.Willioms于1964年设计

1.堆的定义

堆是每个非叶结点值都大于或等于其儿子值（父大于子）的完全二叉树

用数组a[n+1]存储长度为n的堆（a[0]不用）

a[i]≥a[2*i]

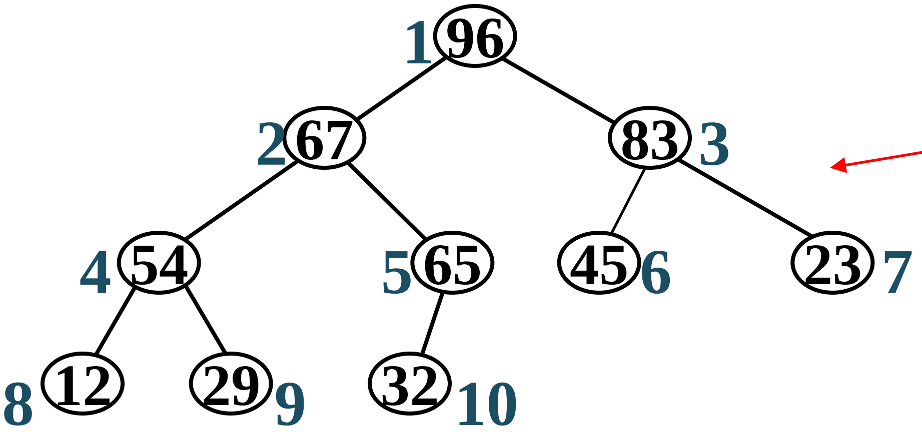
a[i]≥a[2*i+1]　　大根堆

根结点a[1]存储的是最大值　　也可定义小根堆

2. 示例

	0	1	2	3	4	5	6	7	8	9	10
a		96	67	83	54	65	45	23	12	29	32

存储堆的数组



堆的逻辑结构

3．排序阶段的操作方法

步骤1）令j=n;

j是堆的最大下标，即堆的当前尾指针

步骤2）将a[1]与a[j]交换，将最大元换到当前尾

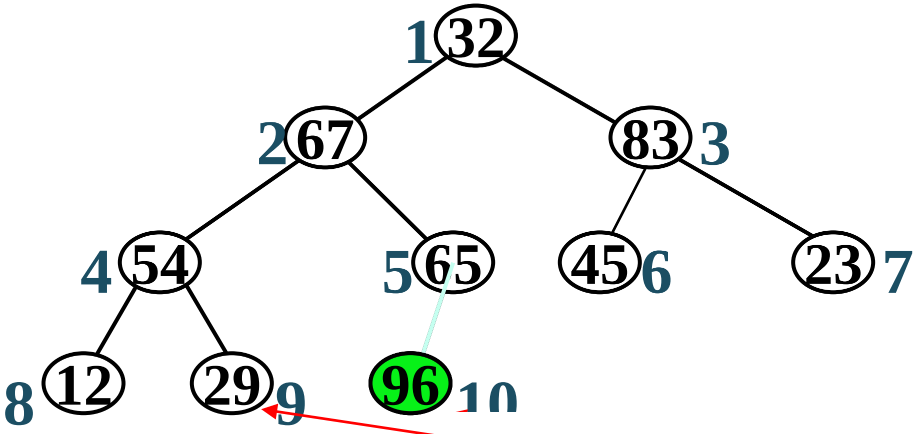
步骤3）j=j-1；使堆的范围缩小

步骤4）重新堆化（是关键步骤）

即调整a[1]~a[j]，使之成为一个新堆

步骤5）若j>1，则转步骤2；否则排序结束

示例（n=10）



步1　j=n=10

步2　a[1]与a[10]交换

步3　j=j-1（j=9）

步4　重新堆化a[1]至a[j]

j

绿色结点脱离了堆

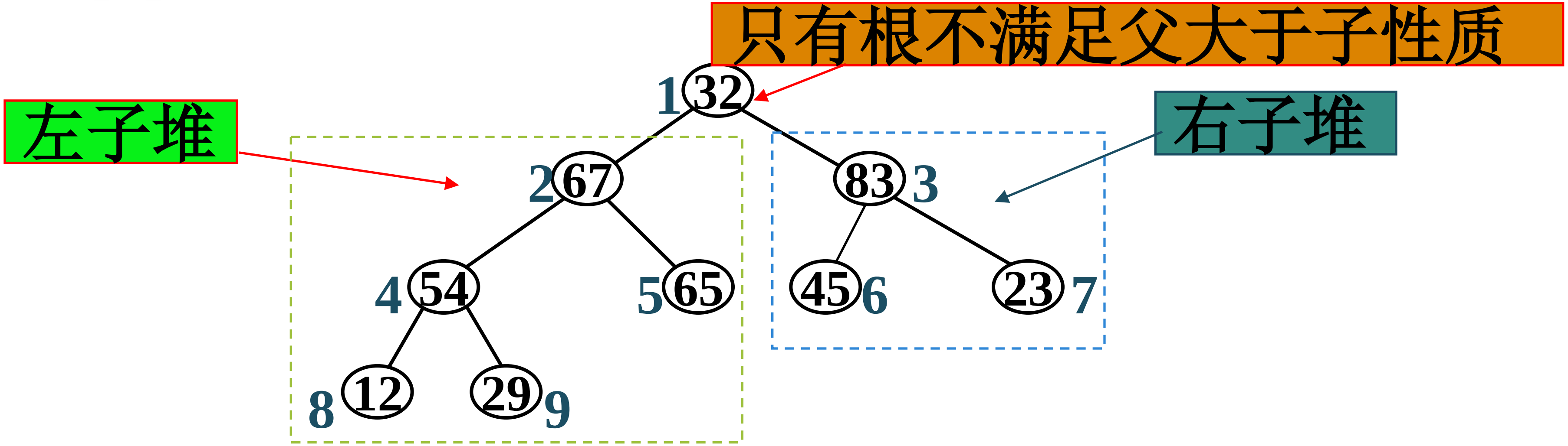
	0	1	2	3	4	5	6	7	8	9	10
a		32	67	83	54	65	45	23	12	29	96

如何进行重新堆化工作？

重新堆化实质——子堆合并

重新堆化范围：a[1]至a[j]（ j是当前堆尾，最大下标）

根a[1]的左右子树满足父大于子性质（子堆）



利用左右子堆合并原理，进行重新堆化

4. 重新堆化步骤

步骤1）i=1

步骤2）若i是叶，或a[i]已满足父大于子的性质，重新堆化结束；否则继续下一步

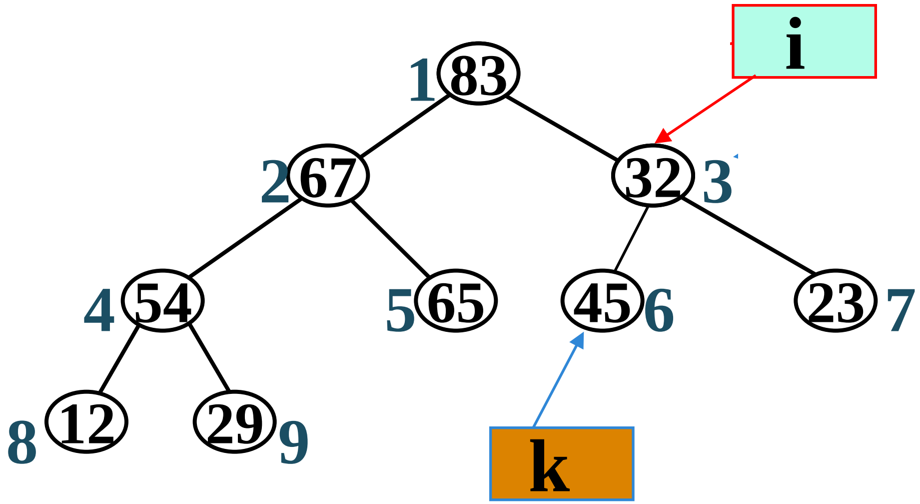
步骤3）找i的“大儿子”k

步骤4）交换a[i]与a[k]的值，使根元素下沉

步骤5）i=k，转步骤2

重新堆化示例

重新堆化范围：a[1]至a[9]，当前根下标i=1



步1 i是叶？

a[i]满足父大于子？

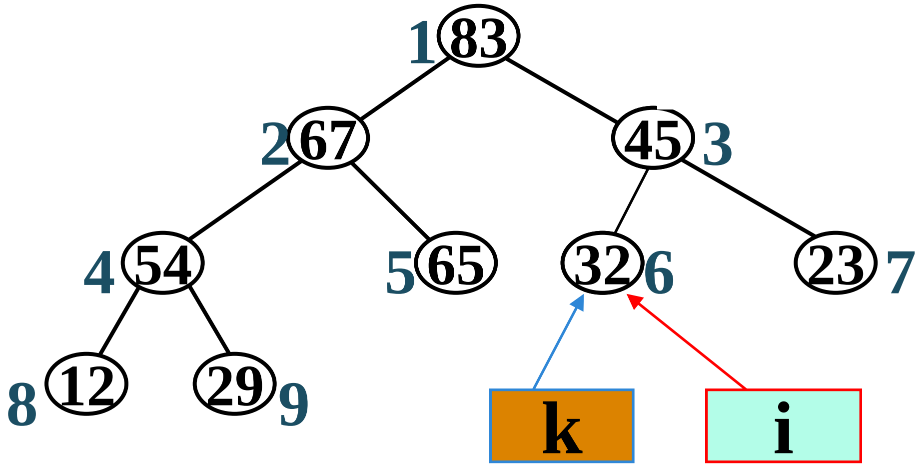
步2 找i的“大儿子”k=3

步3 交换a[1]与a[3]的值

步4 i=k=3，转步1

重新堆化示例

重新堆化范围：a[1]至a[9]。



堆高h≤1+$\log_2 n$
重新堆化时间用量
不超过O(logn)

重新堆化结束

步1 i是叶？
a[i]满足父大于子？

步2 找i的“大儿子”k=6

步3 交换a[3]与a[6]的值

步4 i=k=6，转步1

重新堆化函数

```
void heapify(int a[ ],int i, int j)
{ int k,x;
1.    k=2*i;        // k是i的左儿子
2.     x=a[i];       //将a[i]存入临时变量x中，使a[i]单元空出
3.    while(k<=j)  // 当i不是叶
       {
4.        if(k<j)  //若i有两个儿子时
5.          if(a[k]<a[k+1])  k=k+1;  //k指向i的大儿子
6.        if(x>=a[k])  break; // i的儿子都不大于a[i]下滲结束
7.        a[i]=a[k], i=k, k=2*i;  //大儿子上升, i指向下一数据
       }
8.   a[i]=x;  //原根元素值就位
   }
```

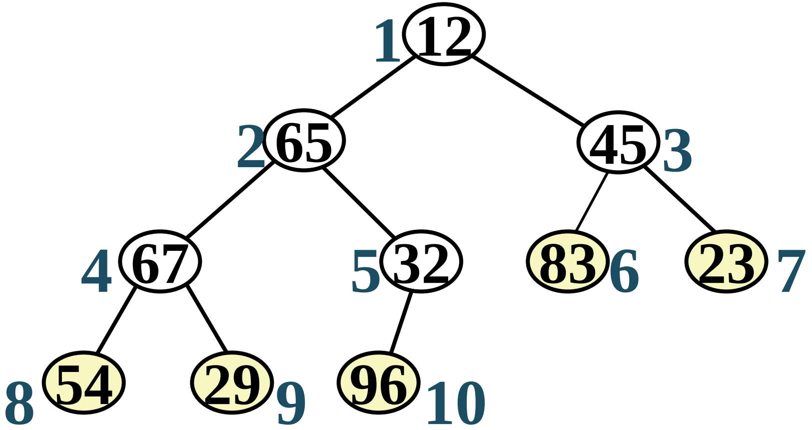
初始堆化步骤和示例

- 利用重新堆化原理
  - 开始时，每个叶结点单独构成子堆
  - 从最下层的非叶结点起，反复进行子堆合并
  - 自底向上的逐步合并越来越大的堆
- 例如，输入数据

12 65 45 67 32 83 23 54 29 96

构造初始堆的示例 （n=10）

for(i=n/2;i>=1;i−−) reheapify(a,i,n);

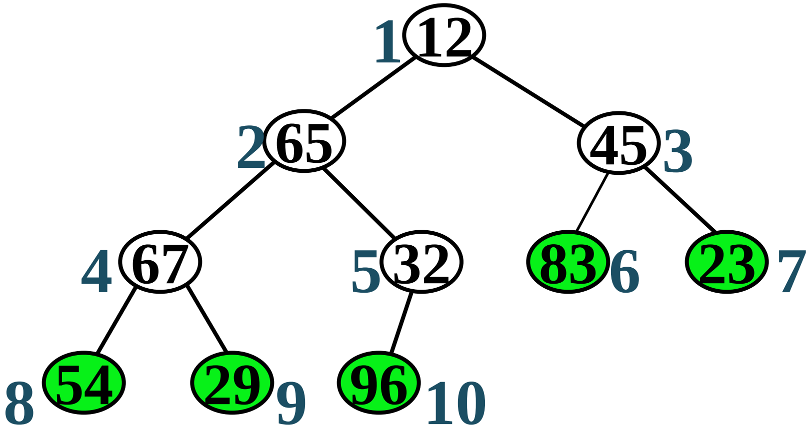


	0	1	2	3	4	5	6	7	8	9	10
a		12	65	45	67	32	83	23	54	29	96

构造初始堆的示例 （n=10）

for(i=n/2;i>=1;i--) reheapify(a,i,n);

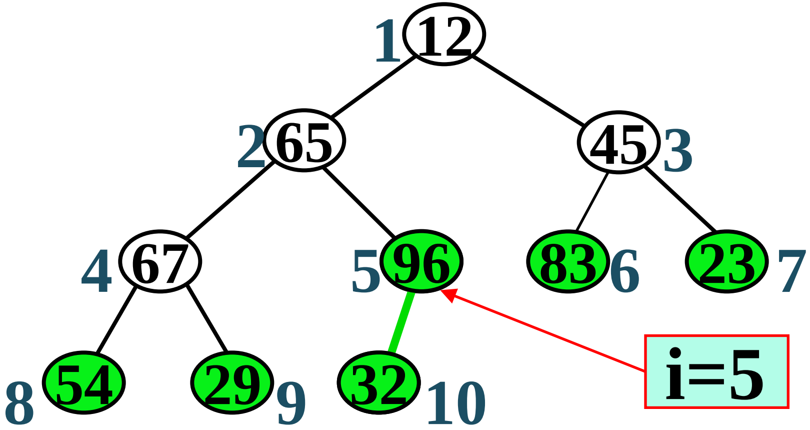
开始时，各叶自成一个子堆（绿色）



	0	1	2	3	4	5	6	7	8	9	10
a		12	65	45	67	32	83	23	54	29	96

构造初始堆的示例 （n=10）

```
for(i=n/2;i>=1;i--) reheapify(a,i,n);
```



各绿色点都是子堆

第一次调用

reheapify(a,5,10);

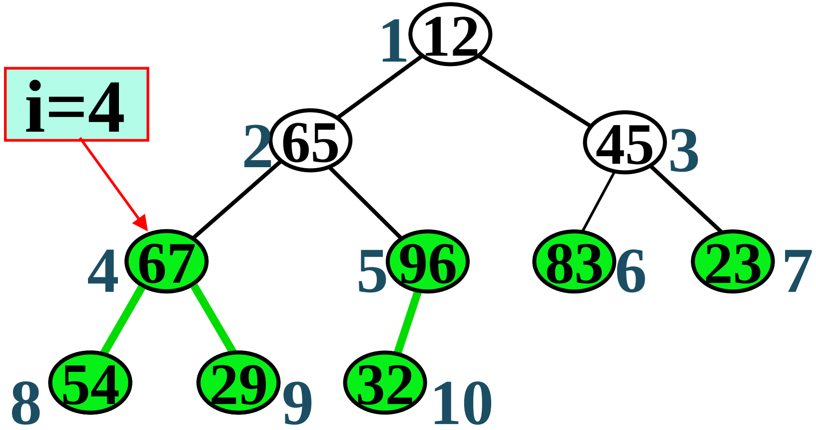
a[5]不大于其儿子a[10]

交换其值

a[5]与a[10]构成一个子堆

构造初始堆的示例 （n=10）

for(i=n/2;i>=1;i--) reheapify(a,i,n);



第二次调用

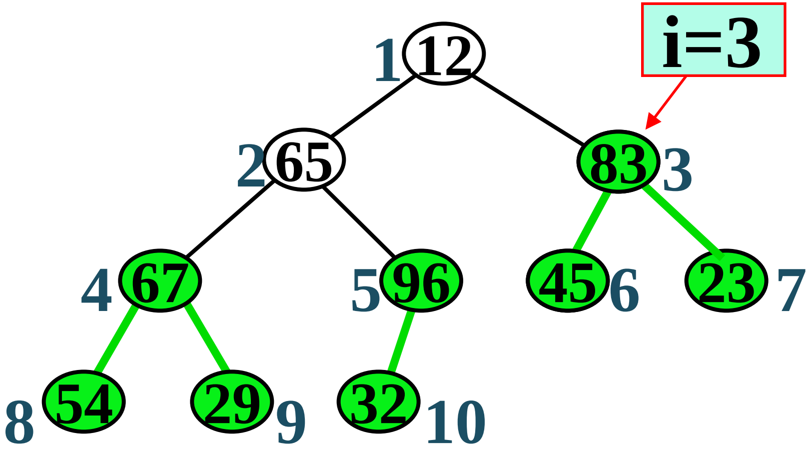
reheapify(a,4,10);

a[4]大于其儿子a[8]和a[9]不交换

a[4]、a[8] 、a[9]构成一个子堆

构造初始堆的示例 （n=10）

for(i=n/2;i>=1;i--) reheapify(a,i,n);



第三次调用

reheapify(a,3,10);

a[3]不大于其儿子a[6]和a[7]

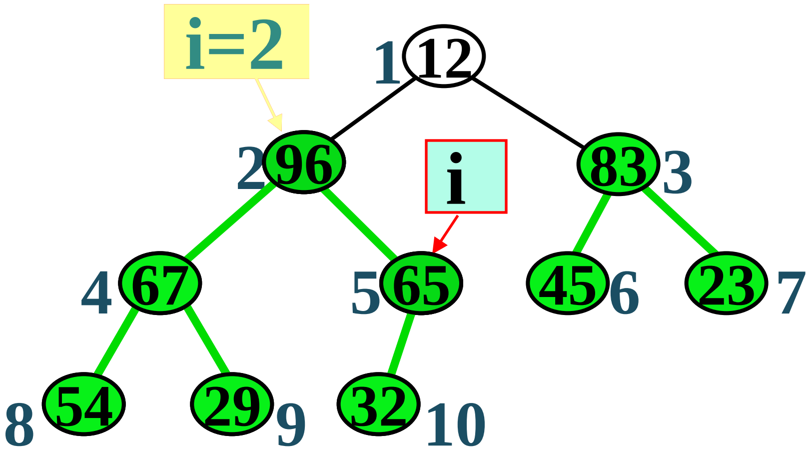
与大儿子a[6]交换

a[3]、a[6] 、a[7]

构成一个子堆

构造初始堆的示例 （n=10）

for(i=n/2;i>=1;i−−) reheapify(a,i,n);



x=65

第四次调用

reheapify(a,2,10);

a[2]不大于其儿子a[4]和a[5]

移走a[2]，大儿子a[5]上升

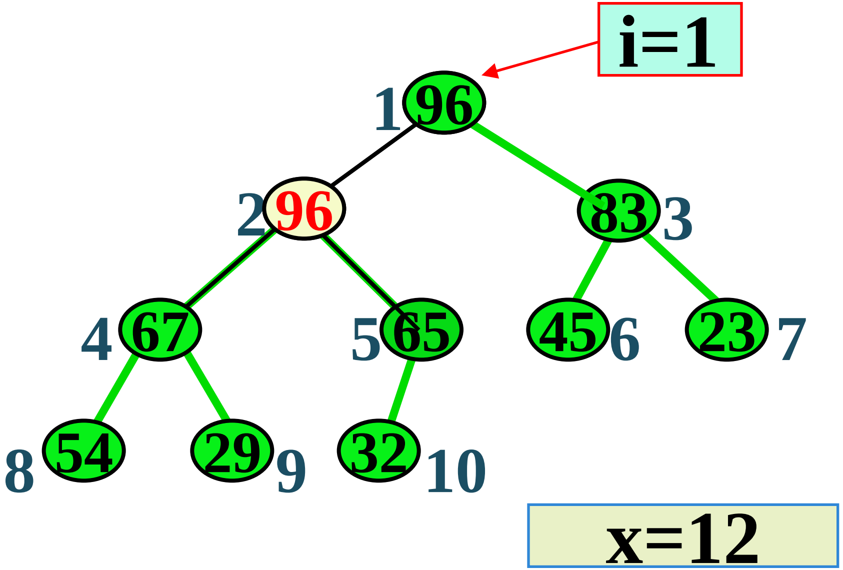
x>其儿子a[10]

x就位于a[5]

构成以a[2]为根的子堆

构造初始堆的示例 （n=10）

for(i=n/2;i>=1;i−−) reheapify(a,i,n);



第五次调用

reheapify(a,1,10);

a[1]不大于其儿子a[2]和a[3]

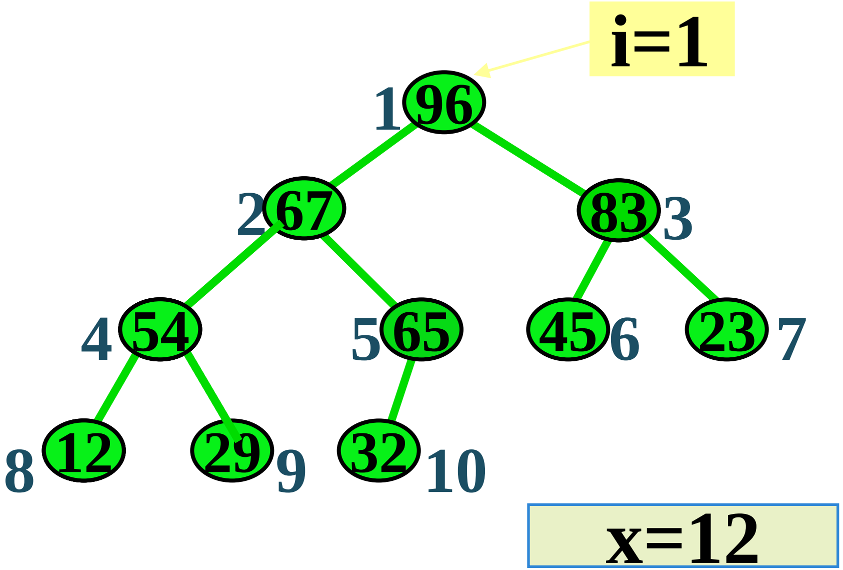
移走a[1]，大儿子a[2]上升

x不大于其儿子a[4]和a[5]

大儿子a[4]上升

构造初始堆的示例 （n=10）

for(i=n/2;i>=1;i--) reheapify(a,i,n);



第五次调用

reheapify(a,1,10);

a[1]不大于其儿子a[2]和a[3]

移走a[1]，大儿子a[2]上升

x不大于其儿子a[4]和a[5]

大儿子a[4]上升

x不大于其儿子a[8]和a[9]

大儿子a[8]上升

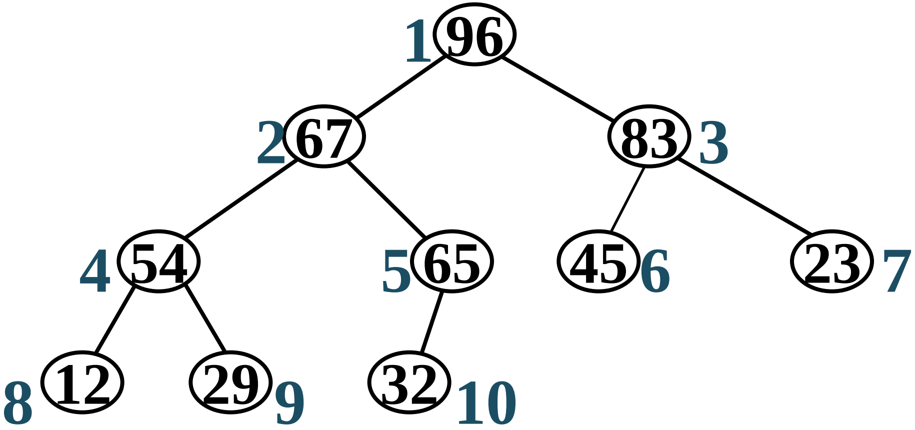
已下降到叶，x就位

完成初始堆化！

堆排序算法

```
//主控函数
void heap_sort(int a[ ],int n)
   {  int i,x;
      for(i=n/2;i>=1;i- -) heapify(a,i,n);  //初始堆化
      for(i=n; i>1; i - -)
        {    x=a[1];  a[1]=a[i];  a[i]=x;
             heapify(a,1,i-1);  //重新堆化
        }
   }
```

为什么堆可以消除冗余比较？



性能分析

（1）初始堆化所需时间不超过O(n)

（2）排序阶段（不含初始堆化）

- 一次重新堆化所需时间不超过O(logn)
- n-1次循环所需时间不超过O(nlogn)

$$T_W(n)=O(n)+O(nlogn)=O(nlogn)$$

各种排序方法比较

排序方法	平均情况	最坏情况	辅助存储	稳定性
直接插入排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
希尔排序	与增量序列有关：$O(n^{3/2})$,$O(n(\log n)^2)$			×
冒泡排序	$O(n^2)$	$O(n^2)$	$O(1)$	√
快速排序	$O(n\log n)$	$O(n^2)$	$O(\log n)$	×
简单选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	×
堆 排 序	$O(n\log n)$	$O(n\log n)$	$O(1)$	×