

本节内容

# 堆排序

王道考研/CSKAOYAN.COM

公众号： 考研发条      一手课程！

## 知识总览

选择排序

简单选择排序

堆排序

重点来了 快认真听



选择排序：每一趟在待排序元素中选取关键字最小（或最大）的元素加入有序子序列

王道考研/CSKAOYAN.COM

## 什么是“堆 (Heap)”?

若  $n$  个关键字序列  $L[1..n]$  满足下面某一条性质，则称为堆 (Heap)：

- ① 若满足： $L(i) \geq L(2i)$  且  $L(i) \geq L(2i+1)$  ( $1 \leq i \leq n/2$ ) —— 大根堆 (大顶堆)
- ② 若满足： $L(i) \leq L(2i)$  且  $L(i) \leq L(2i+1)$  ( $1 \leq i \leq n/2$ ) —— 小根堆 (小顶堆)

大根堆

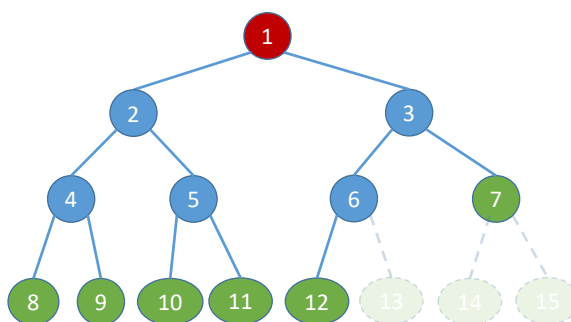


小根堆



王道考研/CSKAOYAN.COM

## 二叉树的顺序存储



几个重要常考的基本操作：

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$
- $i$  所在的层次 ——  $\lceil \log_2(n+1) \rceil$  或  $\lfloor \log_2 n \rfloor + 1$



若完全二叉树中共有  $n$  个结点，则

- 判断  $i$  是否有左孩子？ ——  $2i \leq n$  ?
- 判断  $i$  是否有右孩子？ ——  $2i+1 \leq n$  ?
- 判断  $i$  是否是叶子/分支结点？ ——  $i > \lfloor n/2 \rfloor$  ?



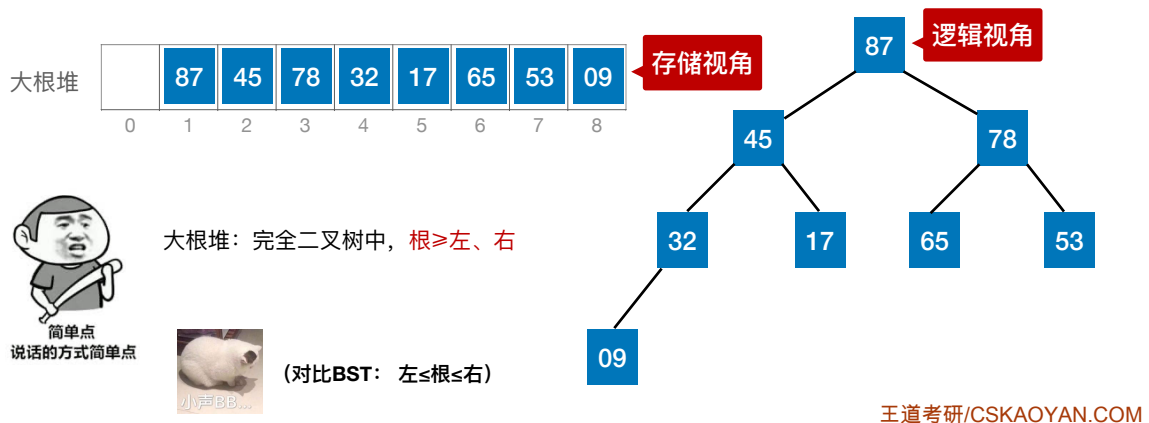
$t[0]$   $t[1]$   $t[2]$  .....

王道考研/CSKAOYAN.COM

## 什么是“堆 (Heap)”?

若  $n$  个关键字序列  $L[1 \dots n]$  满足下面某一条性质，则称为堆 (Heap)：

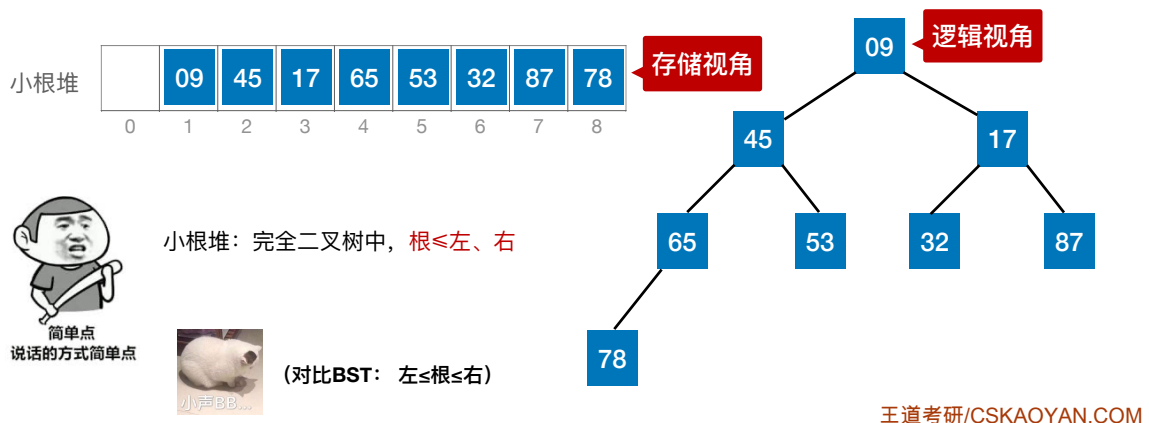
- ① 若满足： $L(i) \geq L(2i)$  且  $L(i) \geq L(2i+1)$  ( $1 \leq i \leq n/2$ ) —— 大根堆 (大顶堆)
- ② 若满足： $L(i) \leq L(2i)$  且  $L(i) \leq L(2i+1)$  ( $1 \leq i \leq n/2$ ) —— 小根堆 (小顶堆)



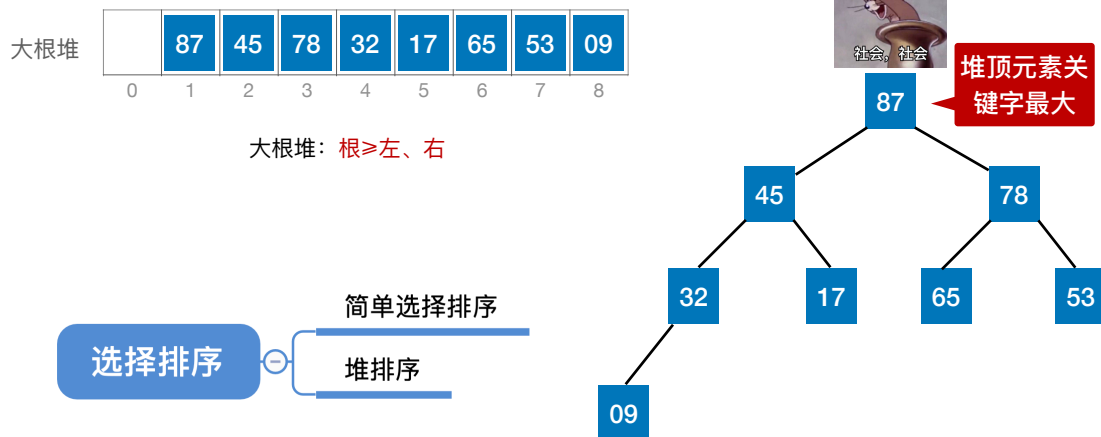
## 什么是“堆 (Heap)”?

若  $n$  个关键字序列  $L[1 \dots n]$  满足下面某一条性质，则称为堆 (Heap)：

- ① 若满足： $L(i) \geq L(2i)$  且  $L(i) \geq L(2i+1)$  ( $1 \leq i \leq n/2$ ) —— 大根堆 (大顶堆)
- ② 若满足： $L(i) \leq L(2i)$  且  $L(i) \leq L(2i+1)$  ( $1 \leq i \leq n/2$ ) —— 小根堆 (小顶堆)



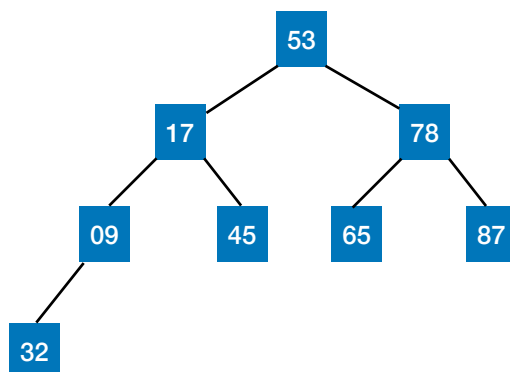
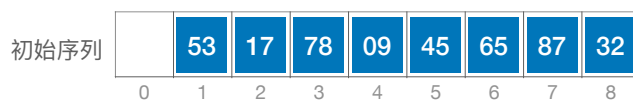
## 如何基于“堆”进行排序?



选择排序: 每一趟在待排序元素中选取关键字最小(或最大)的元素加入有序子序列

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆: 根 $\geq$ 左、右

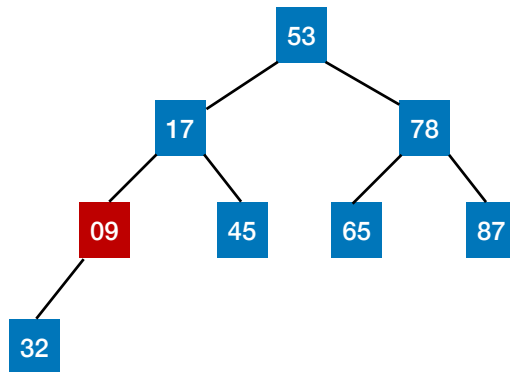
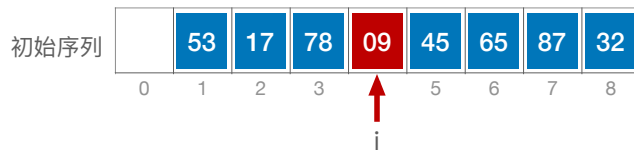
思路: 把所有非终端结点都检查一遍, 是否满足大根堆的要求, 如果不满足, 则进行调整



在顺序存储的完全二叉树中, 非终端结点编号  $i \leq \lfloor n/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

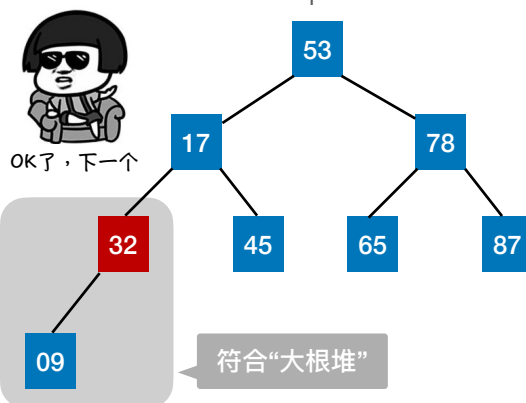
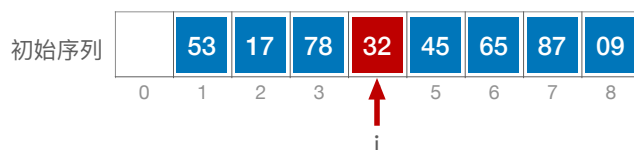
思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

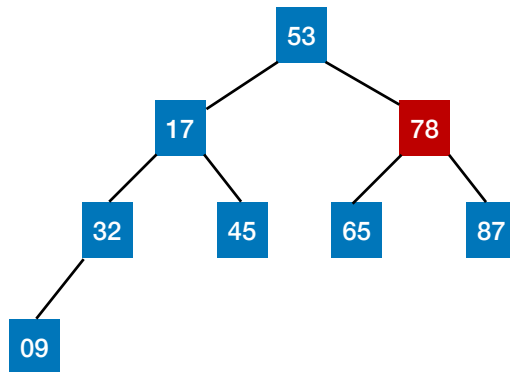
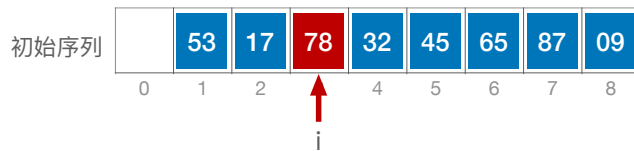
思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

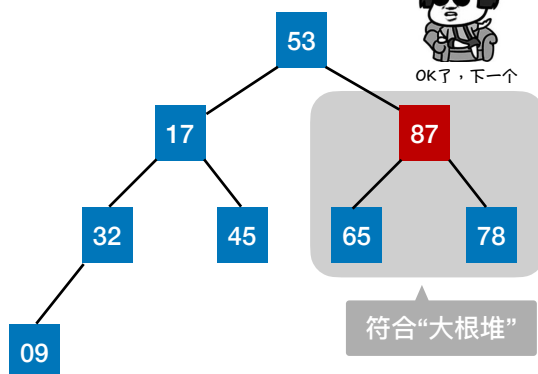
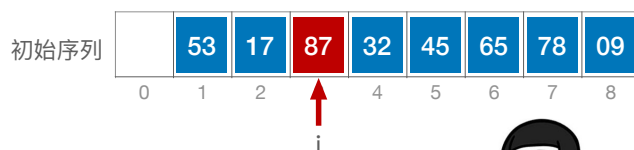
思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

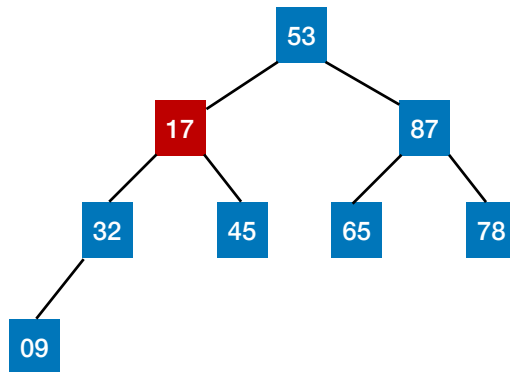
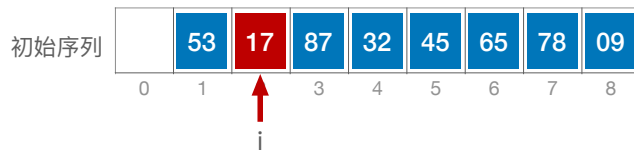
思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

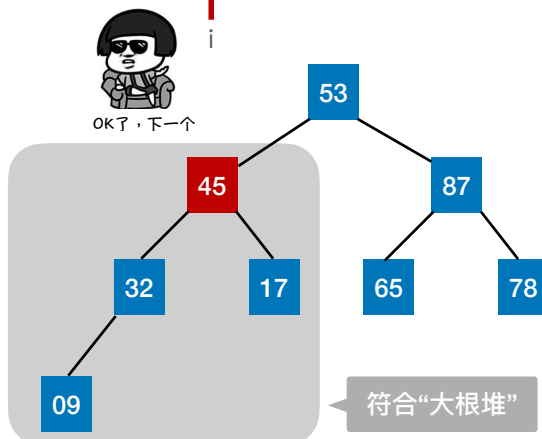
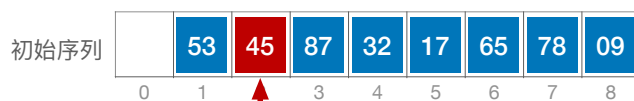
思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

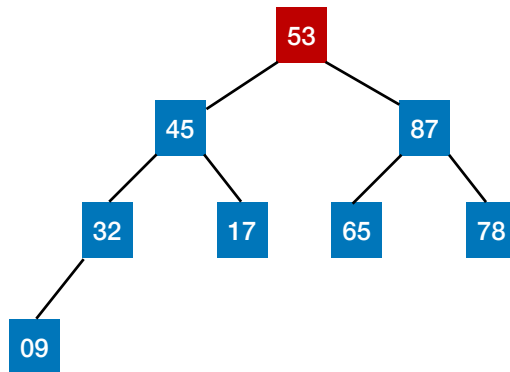
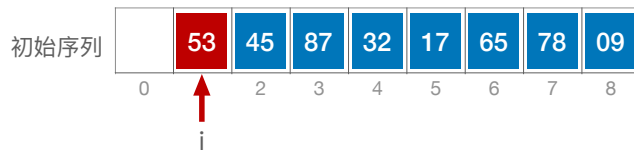
思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

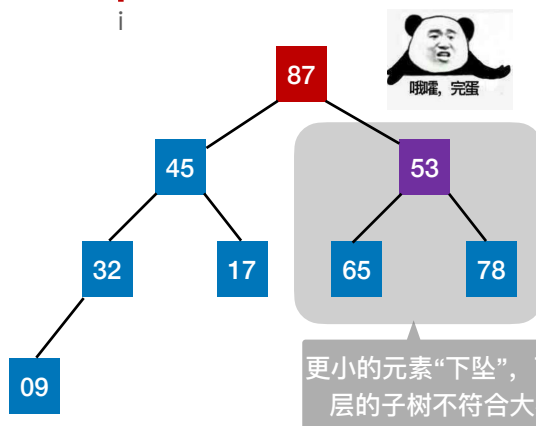
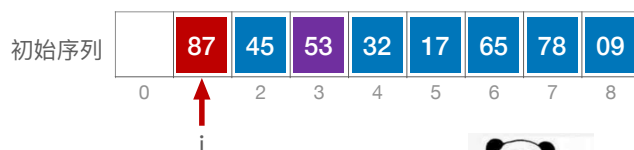
思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- i 的左孩子 ——  $2i$
- i 的右孩子 ——  $2i+1$
- i 的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

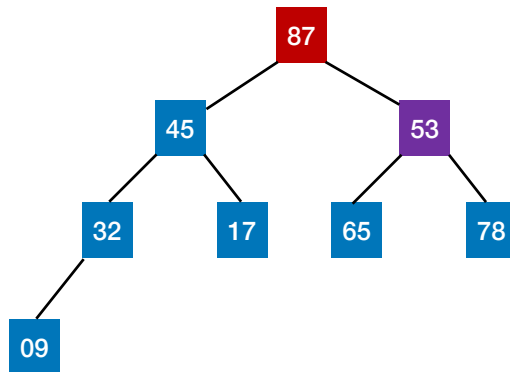
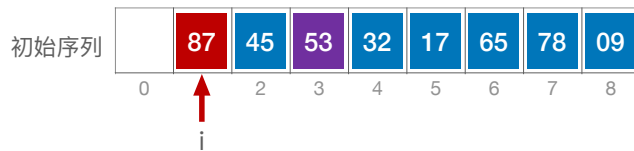
检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

- i 的左孩子 ——  $2i$
- i 的右孩子 ——  $2i+1$
- i 的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM



## 建立大根堆



大根堆：根 $\geq$ 左、右

思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

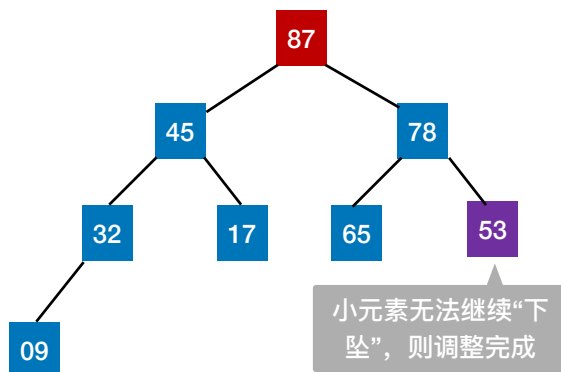
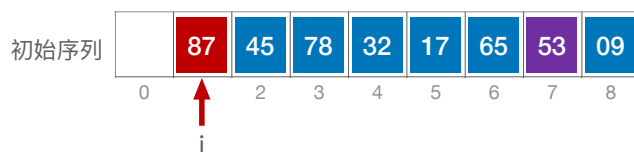
检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

若元素互换破坏了下一级的堆，则采用相同的方法继续往下调整（小元素不断“下坠”）

- i 的左孩子 ——  $2i$
- i 的右孩子 ——  $2i+1$
- i 的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整

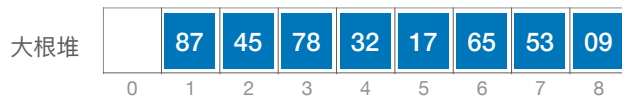
检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

若元素互换破坏了下一级的堆，则采用相同的方法继续往下调整（小元素不断“下坠”）

- i 的左孩子 ——  $2i$
- i 的右孩子 ——  $2i+1$
- i 的父节点 ——  $\lfloor i/2 \rfloor$

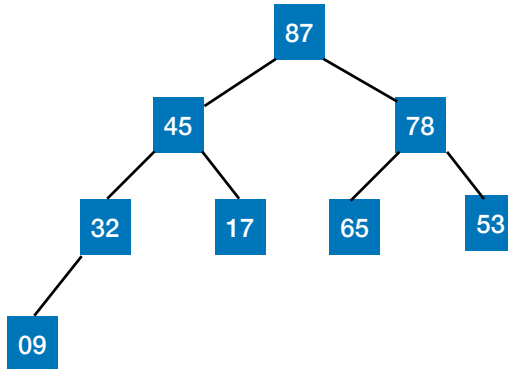
王道考研/CSKAOYAN.COM

## 建立大根堆



大根堆：根 $\geq$ 左、右

思路：把所有非终端结点都检查一遍，是否满足大根堆的要求，如果不满足，则进行调整



检查当前结点是否满足 根 $\geq$ 左、右  
若不满足，将当前结点与更大的一个孩子互换

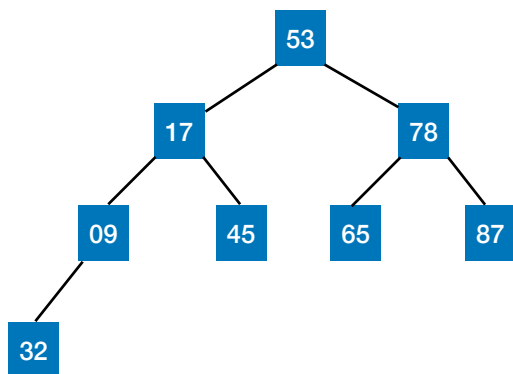
若元素互换破坏了下一级的堆，则采用相同的方法继续往下调整（小元素不断“下坠”）

- $i$  的左孩子 ——  $2i$
- $i$  的右孩子 ——  $2i+1$
- $i$  的父节点 ——  $\lfloor i/2 \rfloor$

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）

初始序列



//建立大根堆

```
void BuildMaxHeap(int A[],int len){  
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点  
        HeadAdjust(A,i,len);  
}
```

//将以  $k$  为根的子树调整为大根堆

```
void HeadAdjust(int A[],int k,int len){  
    A[0]=A[k]; //A[0]暂存子树的根结点  
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选  
        if(i<len&&A[i]<A[i+1])  
            i++; //取key较大的子结点的下标  
        if(A[0]>=A[i]) break; //筛选结束  
        else{  
            A[k]=A[i]; //将A[i]调整到双亲结点上  
            k=i; //修改k值，以便继续向下筛选  
        }  
    }  
    A[k]=A[0]; //被筛选结点的值放入最终位置  
}
```

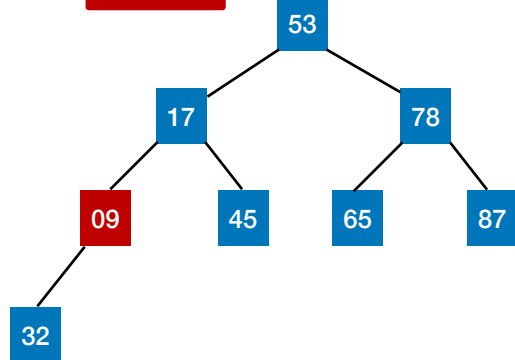
王道考研/CSKAOYAN.COM

## 建立大根堆（代码）

初始序列

	53	17	78	09	45	65	87	32
0	1	2	3	4	5	6	7	8

从最底层的分支  
结点开始调整



//建立大根堆

```
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}
```

//将以 k 为根的子树调整为大根堆

```
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值，以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

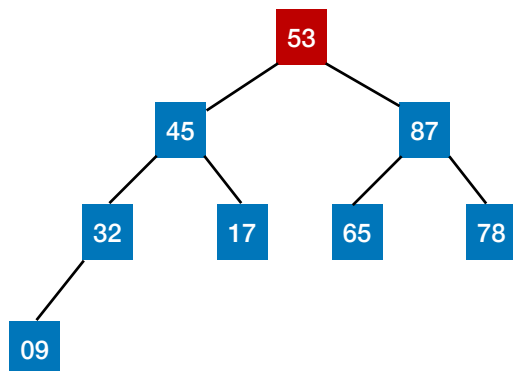
王道考研/CSKAOYAN.COM

## 建立大根堆（代码）

53	53	45	87	32	17	65	78	09
0	1	2	3	4	5	6	7	8

k

i



//建立大根堆

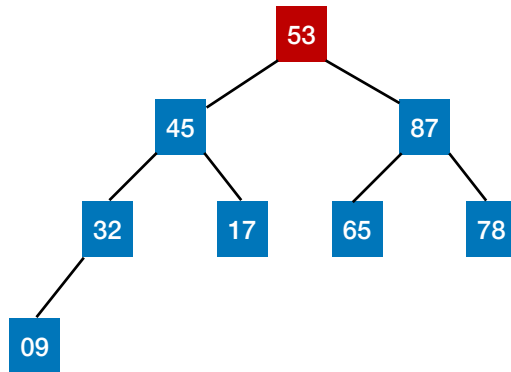
```
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}
```

//将以 k 为根的子树调整为大根堆

```
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值，以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）



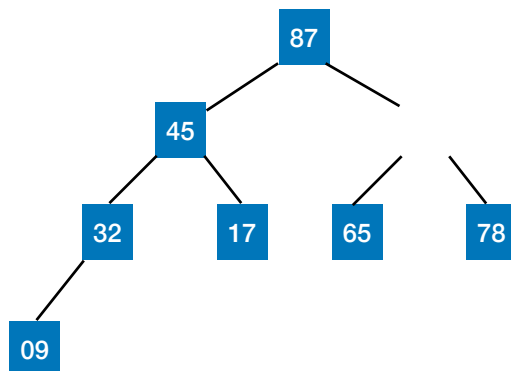
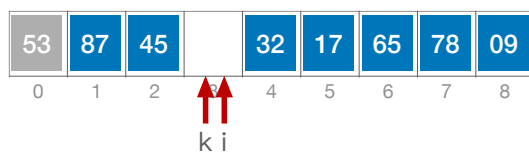
```

//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值，以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
    
```

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）



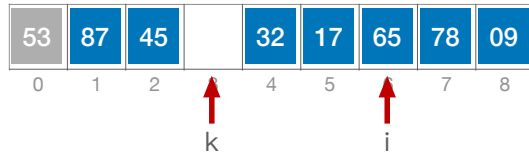
```

//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值，以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
    
```

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）



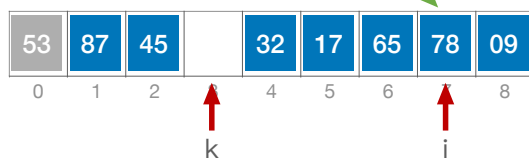
```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）

右孩子更大

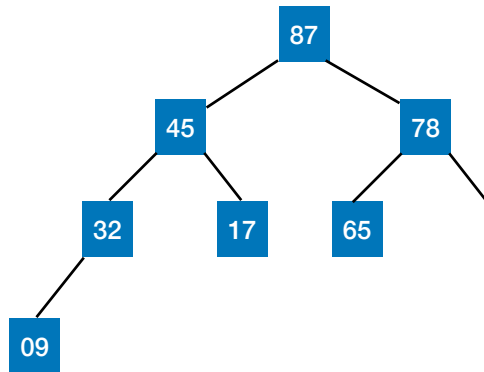
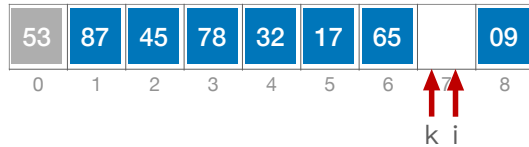


```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）

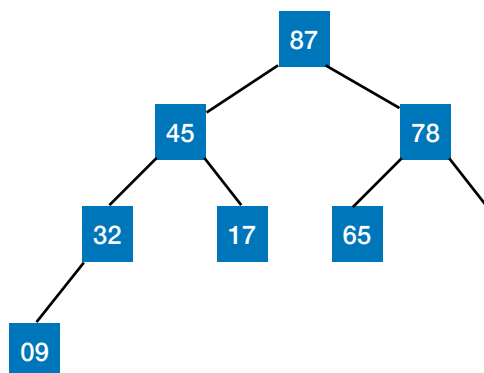
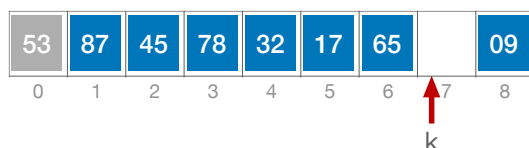


```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）

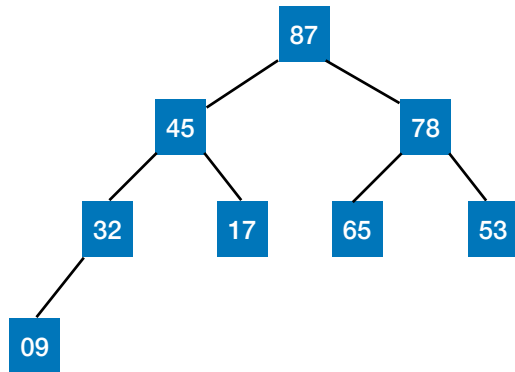
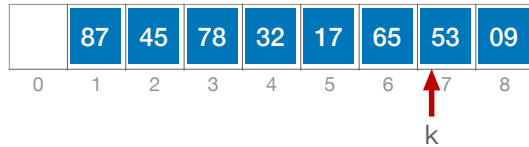


```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

王道考研/CSKAOYAN.COM

## 建立大根堆（代码）



```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

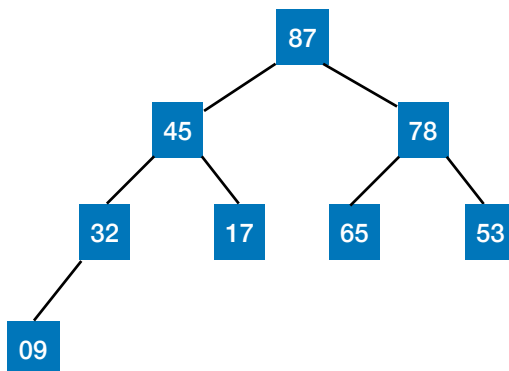
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

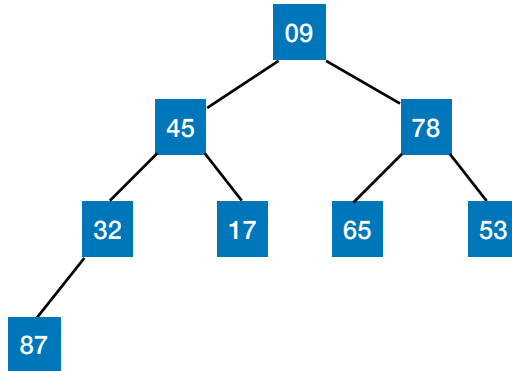
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

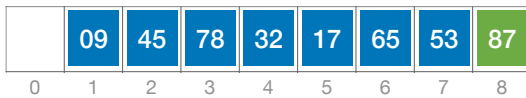


堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

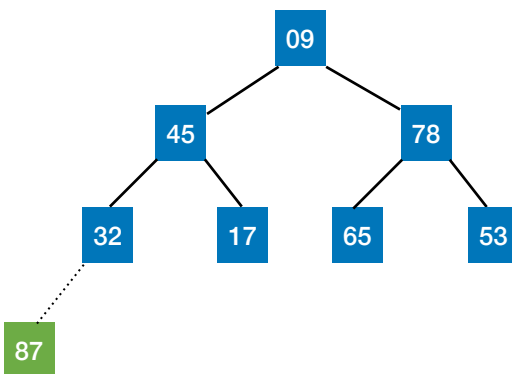
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列



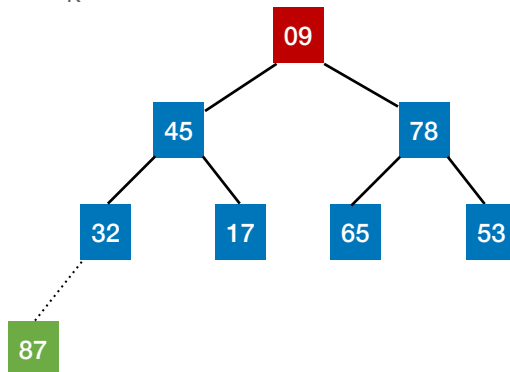
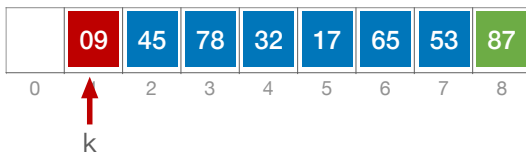
堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

王道考研/CSKAOYAN.COM



## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

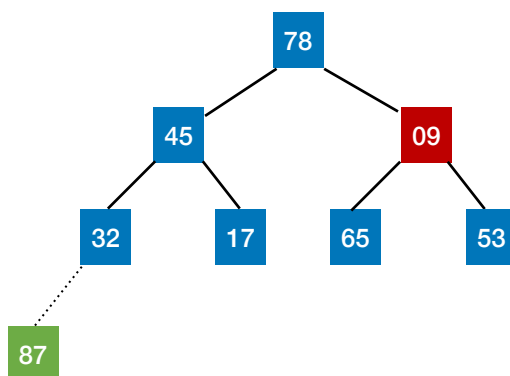
`void HeadAdjust(int A[], int k, int len)`

len=7

王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=7

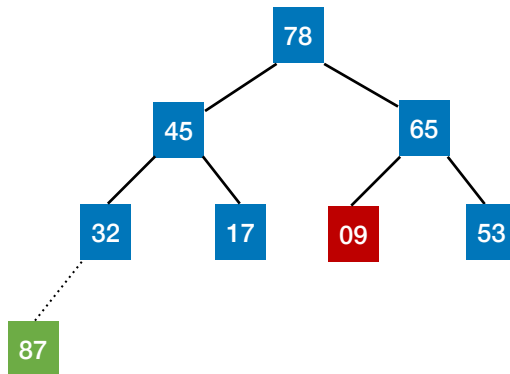
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=7

王道考研/CSKAOYAN.COM

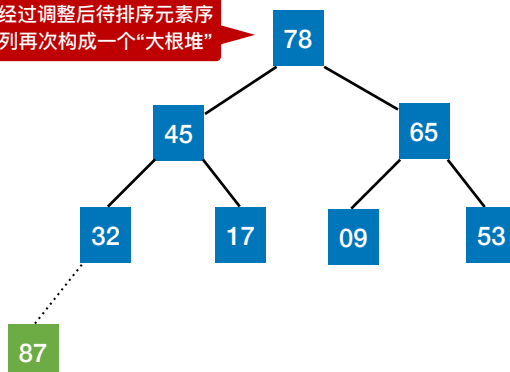
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

经过调整后待排序元素序列再次构成一个“大根堆”



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=7

王道考研/CSKAOYAN.COM

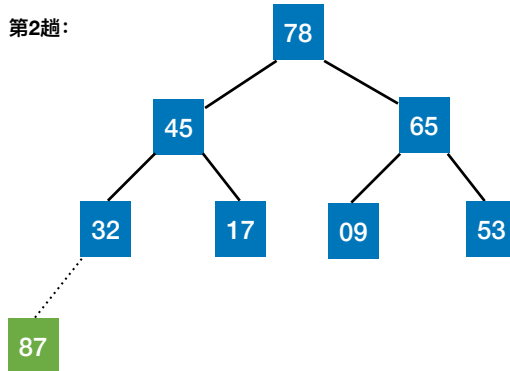
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第2趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

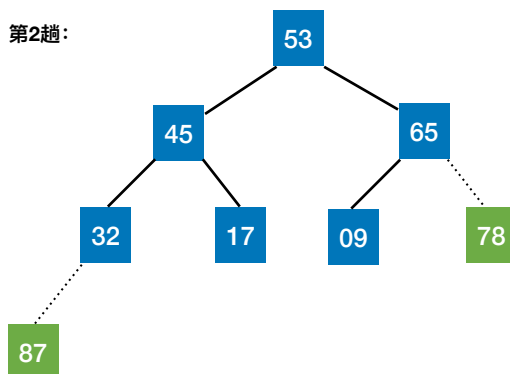
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第2趟：



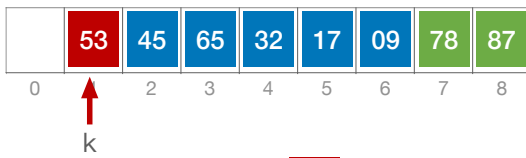
堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

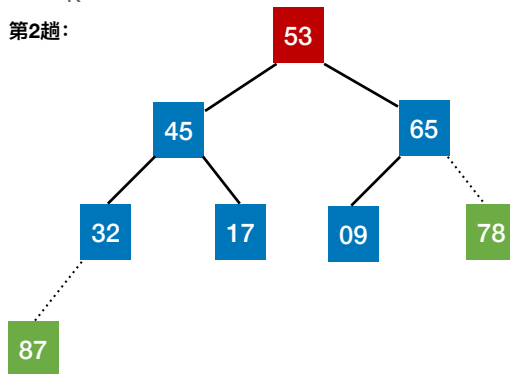
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



第2趟:



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

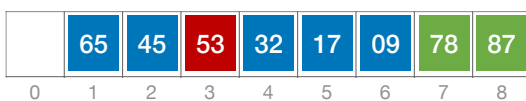
`void HeadAdjust(int A[], int k, int len)`

len=6

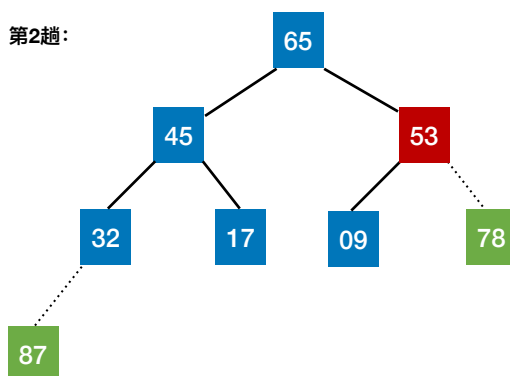
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



第2趟:



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=6

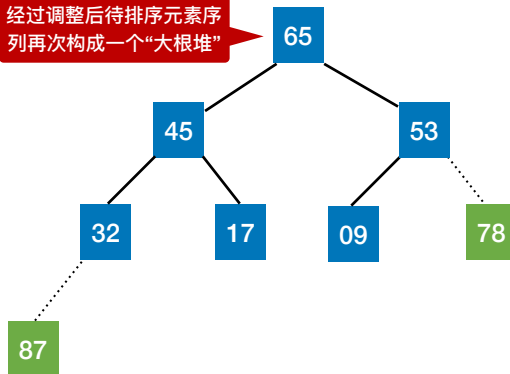
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



经过调整后待排序元素序列再次构成一个“大根堆”



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

`len=6`

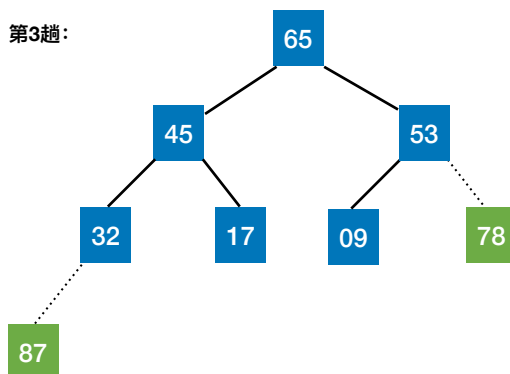
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



第3趟：



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

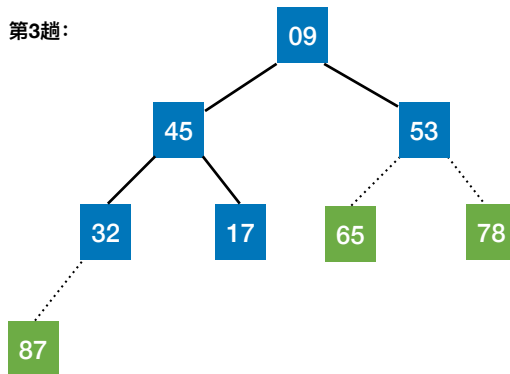
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第3趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

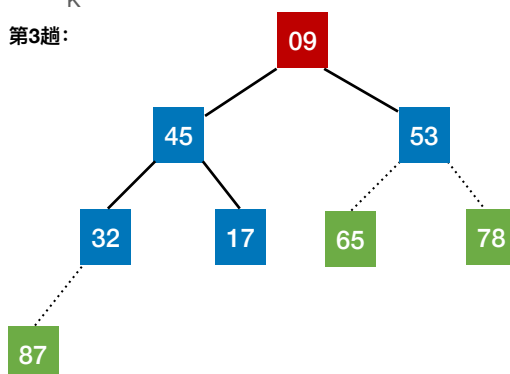
## 基于大根堆进行排序

大根堆



k

第3趟：



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以 k 为根的子树调整为大根堆

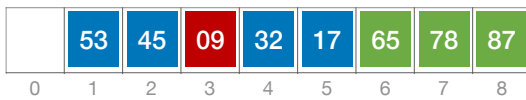
`void HeadAdjust(int A[], int k, int len)`

len=5

王道考研/CSKAOYAN.COM

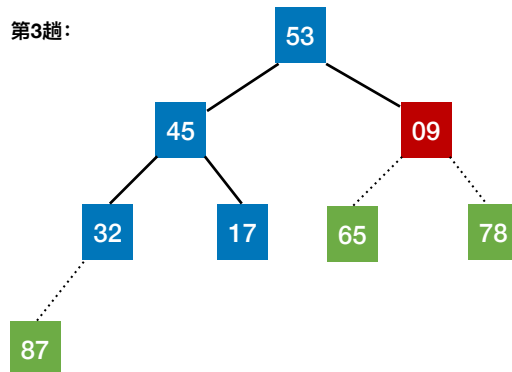
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第3趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

`len=5`

王道考研/CSKAOYAN.COM

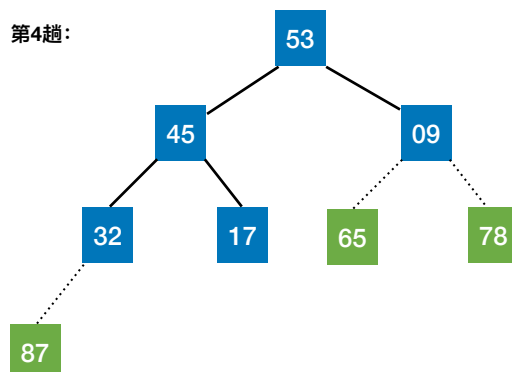
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第4趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

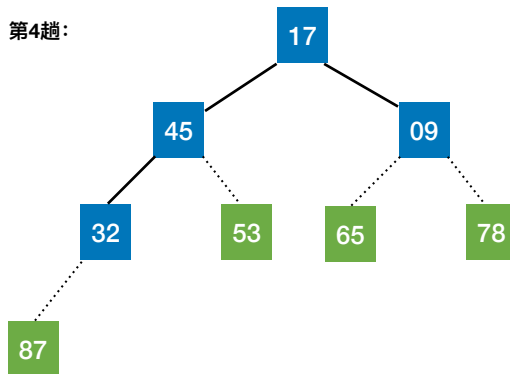
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第4趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

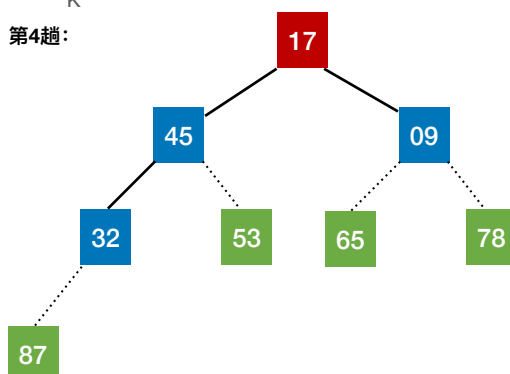
## 基于大根堆进行排序

大根堆



k

第4趟：



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以 k 为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=4

王道考研/CSKAOYAN.COM



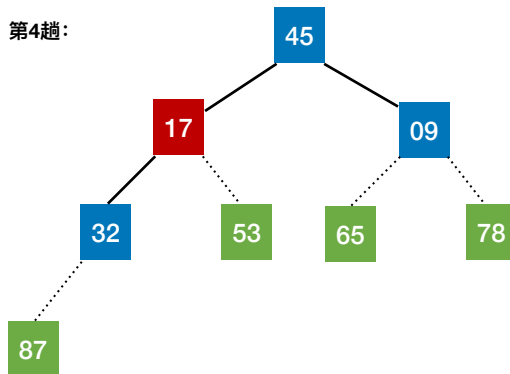
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第4趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

```
void HeadAdjust(int A[], int k, int len)
```

len=4

王道考研/CSKAOYAN.COM

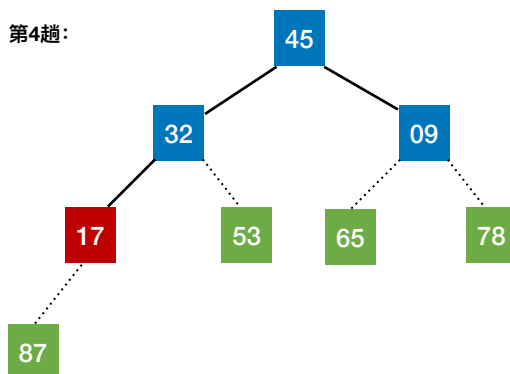
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第4趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

```
void HeadAdjust(int A[], int k, int len)
```

len=4

王道考研/CSKAOYAN.COM

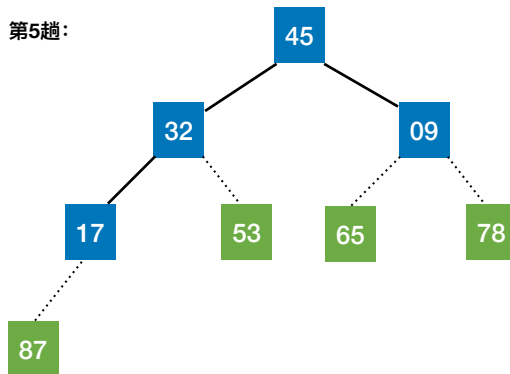
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第5趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

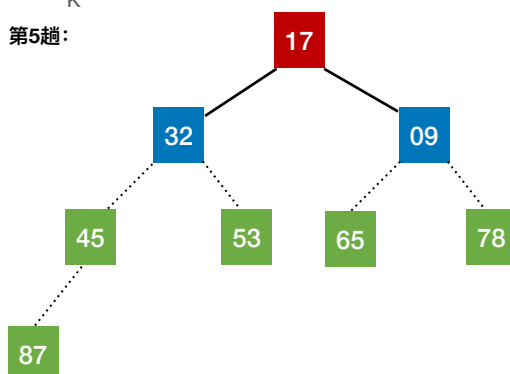
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第5趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

`len=3`

王道考研/CSKAOYAN.COM

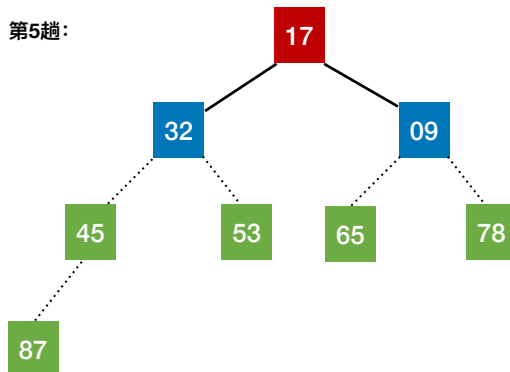
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第5趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=3

王道考研/CSKAOYAN.COM

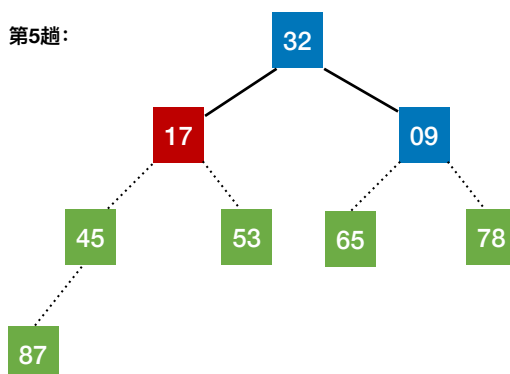
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第5趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=3

王道考研/CSKAOYAN.COM

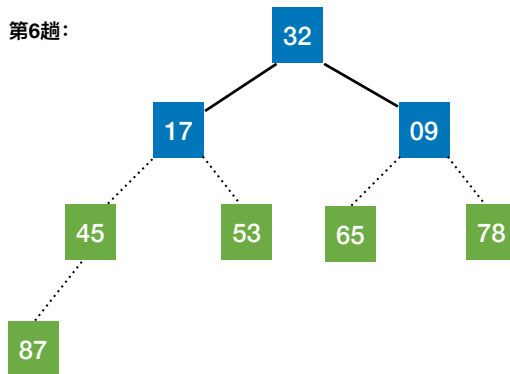
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第6趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

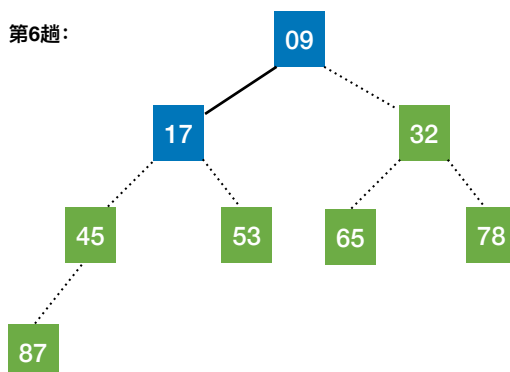
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第6趟：



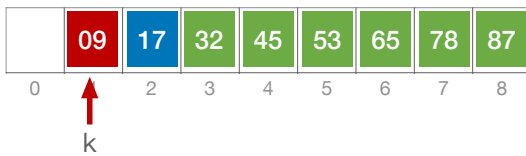
堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

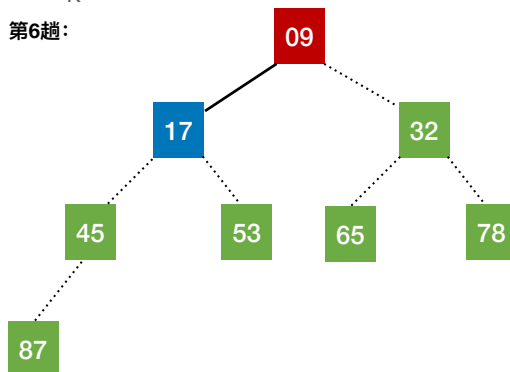
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



第6趟:



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=2

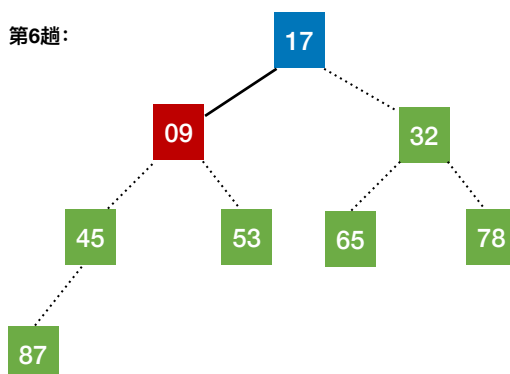
王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

大根堆



第6趟:



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

//将以  $k$  为根的子树调整为大根堆

`void HeadAdjust(int A[], int k, int len)`

len=2

王道考研/CSKAOYAN.COM

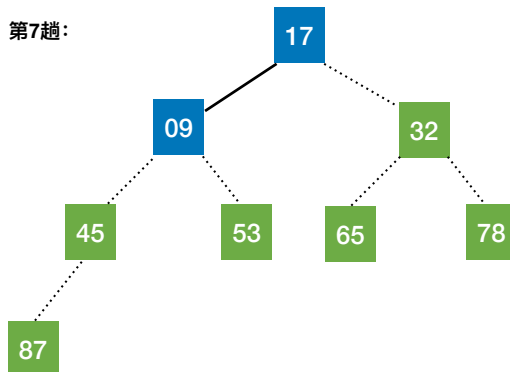
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第7趟：



堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

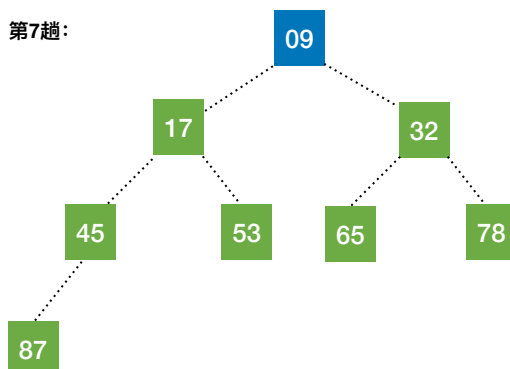
## 基于大根堆进行排序

大根堆



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

第7趟：



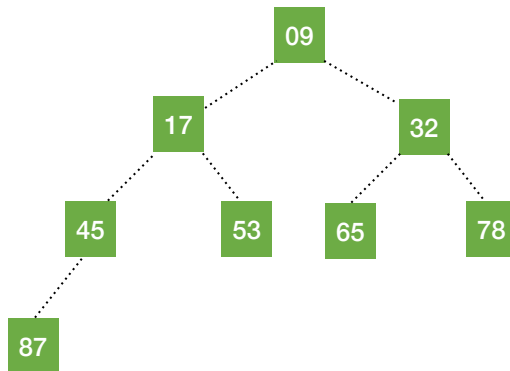
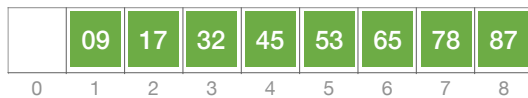
堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

## 基于大根堆进行排序

$n-1$ 趟 处理之后:



选择排序：每一趟在待排序元素中选取关键字最大的元素加入有序子序列

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

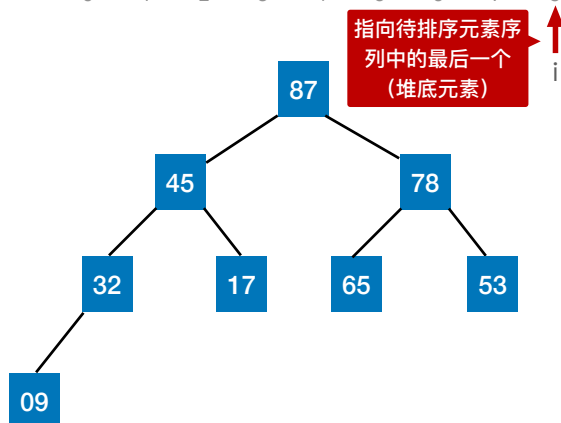
并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

注意：基于“大根堆”的堆排序得到“递增序列”

王道考研/CSKAOYAN.COM

## 基于大根堆进行排序（代码）

大根堆



//建立大根堆

```
void BuildMaxHeap(int A[],int len)
```

//将以  $k$  为根的子树调整为大根堆

```
void HeadAdjust(int A[],int k,int len)
```

//堆排序的完整逻辑

```
void HeapSort(int A[],int len){
    BuildMaxHeap(A, len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}
```

堆排序：每一趟将堆顶元素加入有序子序列（与待排序序列中的最后一个元素交换）

并将待排序元素序列再次调整为大根堆（小元素不断“下坠”）

王道考研/CSKAOYAN.COM

## 算法效率分析

```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--){ //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
    }
}

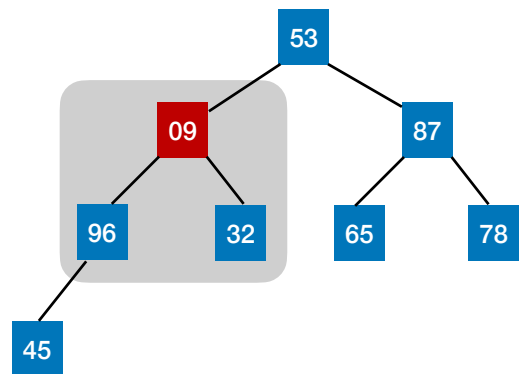
//堆排序的完整逻辑
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}

//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

王道考研/CSKAOYAN.COM

## 算法效率分析

```
//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```



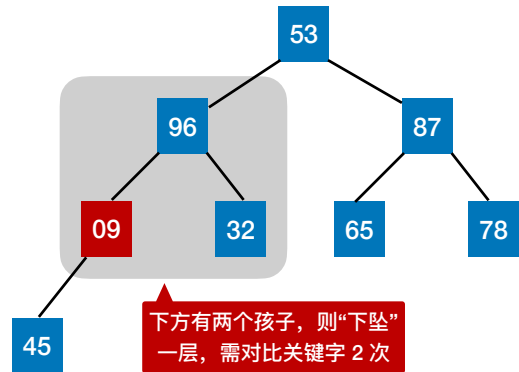
王道考研/CSKAOYAN.COM



## 算法效率分析

//将以  $k$  为根的子树调整为大根堆

```
void HeadAdjust(int A[], int k, int len){
    A[0]=A[k];           //A[0]暂存子树的根结点
    for(int i=2*k; i<=len; i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++;           //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i];     //将A[i]调整到双亲结点上
            k=i;           //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0];           //被筛选结点的值放入最终位置
}
```

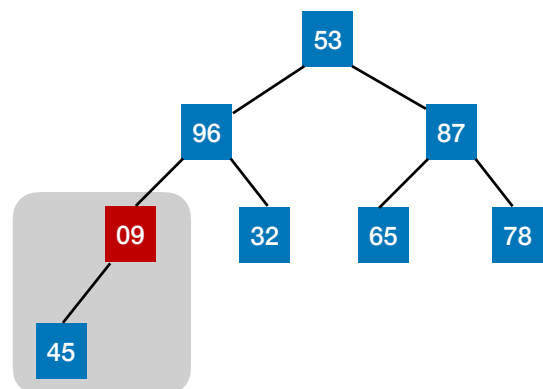


王道考研/CSKAOYAN.COM

## 算法效率分析

//将以  $k$  为根的子树调整为大根堆

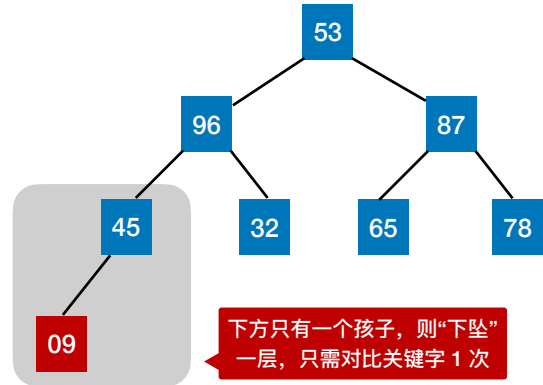
```
void HeadAdjust(int A[], int k, int len){
    A[0]=A[k];           //A[0]暂存子树的根结点
    for(int i=2*k; i<=len; i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++;           //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i];     //将A[i]调整到双亲结点上
            k=i;           //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0];           //被筛选结点的值放入最终位置
}
```



王道考研/CSKAOYAN.COM

## 算法效率分析

```
//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k];           //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++;           //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i];      //将A[i]调整到双亲结点上
            k=i;            //修改k值,以便继续向下筛选
        }
    }
    A[k]=A[0];           //被筛选结点的值放入最终位置
}
```



结论：一个结点，每“下坠”一层，最多只需对比关键字2次

若树高为h，某结点在第i层，则将这个结点向下调整最多只需要“下坠”h-i层，关键字对比次数不超过2(h-i)

王道考研/CSKAOYAN.COM

## 算法效率分析

结论：一个结点，每“下坠”一层，最多只需对比关键字2次

若树高为h，某结点在第i层，则将这个结点向下调整最多只需要“下坠”h-i层，关键字对比次数不超过2(h-i)

n个结点的完全二叉树树高  $h = \lfloor \log_2 n \rfloor + 1$

第i层最多有  $2^{i-1}$  个结点，而只有第1 ~ (h-1)层的结点才有可能需要“下坠”调整

将整棵树调整为大根堆，关键字对比次数不超过  $\sum_{i=h-1}^1 2^{i-1} 2(h-i) = \sum_{i=h-1}^1 2^i (h-i) = \sum_{j=1}^{h-1} 2^{h-j} j \leq 2n \sum_{j=1}^{h-1} \frac{j}{2^j} \leq 4n$

差比数列求和  
(错位相减法)

求和结果小于2

建堆的过程，关键字对比次数不超过  $4n$ ，建堆时间复杂度  $= O(n)$

王道考研/CSKAOYAN.COM

## 算法效率分析

```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}
```

//堆排序的完整逻辑

```
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}
```

$O(n)$

总共需要 $n-1$ 趟，每一趟交换后  
都需要将根节点“下坠”调整

根节点最多“下坠” $h-1$ 层，每下坠一层

而每“下坠”一层，最多只需对比关键字2次，因此每一趟排序复杂度不超过  $O(h) = O(\log_2 n)$

共 $n-1$ 趟，总的时间复杂度 =  $O(n\log_2 n)$

```
//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值，以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

王道考研/CSKAOYAN.COM

## 算法效率分析

```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}
```

//堆排序的完整逻辑

```
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}
```

$O(n)$

$O(n\log_2 n)$

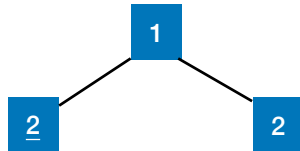
堆排序的时间复杂度 =  $O(n) + O(n\log_2 n) = O(n\log_2 n)$

堆排序的空间复杂度 =  $O(1)$

王道考研/CSKAOYAN.COM

## 稳定性

初始序列:



初始化大根堆

```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

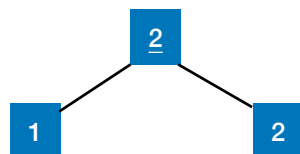
//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值, 以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

注意: 若左右孩子一样大, 则优先和左孩子交换

王道考研/CSKAOYAN.COM

## 稳定性

大根堆: 根 $\geq$ 左、右



```
//建立大根堆
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--) //从后往前调整所有非终端结点
        HeadAdjust(A,i,len);
}

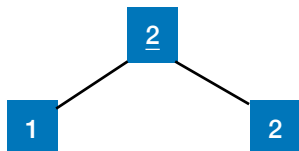
//将以 k 为根的子树调整为大根堆
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k]; //A[0]暂存子树的根结点
    for(int i=2*k;i<=len;i*=2){ //沿key较大的子结点向下筛选
        if(i<len&&A[i]<A[i+1])
            i++; //取key较大的子结点的下标
        if(A[0]>=A[i]) break; //筛选结束
        else{
            A[k]=A[i]; //将A[i]调整到双亲结点上
            k=i; //修改k值, 以便继续向下筛选
        }
    }
    A[k]=A[0]; //被筛选结点的值放入最终位置
}
```

注意: 若左右孩子一样大, 则优先和左孩子交换

王道考研/CSKAOYAN.COM

## 稳定性

大根堆：根 $\geq$ 左、右

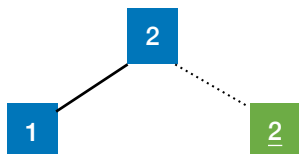


```
//堆排序的完整逻辑
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}
```

王道考研/CSKAOYAN.COM

## 稳定性

大根堆：根 $\geq$ 左、右

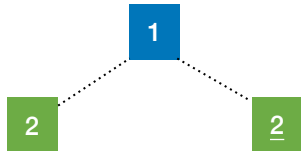


```
//堆排序的完整逻辑
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}
```

王道考研/CSKAOYAN.COM

## 稳定性

大根堆：根 $\geq$ 左、右

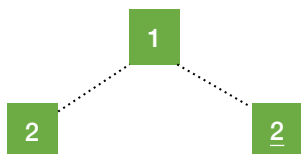


```
//堆排序的完整逻辑
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}
```

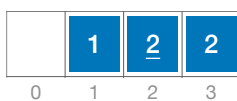
王道考研/CSKAOYAN.COM

## 稳定性

排序结果：



初始序列：



```
//堆排序的完整逻辑
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len); //初始建堆
    for(int i=len;i>1;i--){ //n-1趟的交换和建堆过程
        swap(A[i],A[1]); //堆顶元素和堆底元素交换
        HeadAdjust(A,1,i-1); //把剩余的待排序元素整理成堆
    }
}
```

结论：堆排序是~~不~~稳定的

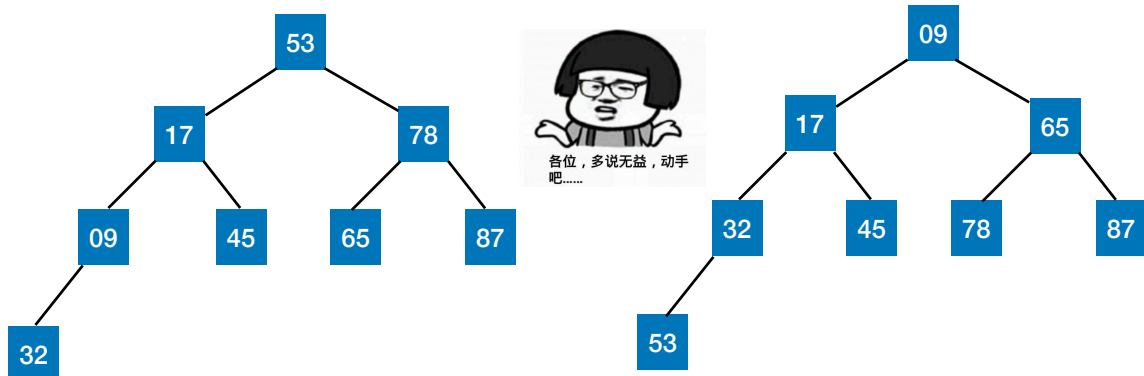
王道考研/CSKAOYAN.COM

## 截屏练习：基于“小根堆”如何建堆、排序？

初始序列



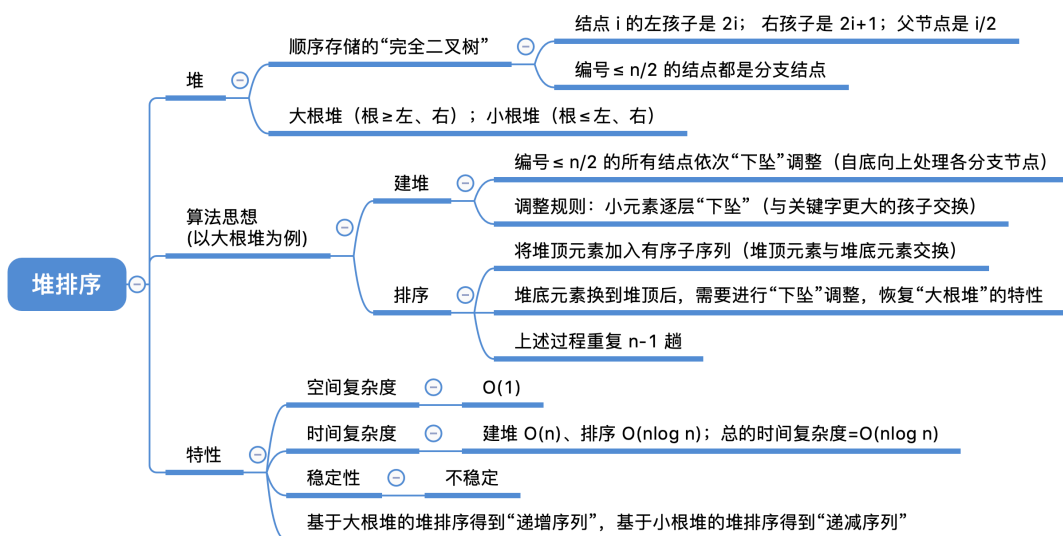
初始化小根堆



注意：基于“小根堆”的堆排序得到“递减序列”

王道考研/CSKAOYAN.COM

## 知识回顾与重要考点



王道考研/CSKAOYAN.COM



@王道论坛



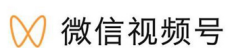
@王道计算机考研备考  
@王道咸鱼老师-计算机考研  
@王道楼楼老师-计算机考研



@王道计算机考研



@王道计算机考研



@王道计算机考研



@王道在线