# 4.2 树的应用

## 4.2.4 堆排序

### 堆的定义

堆是满足下列性质的数列{$r_1, r_2, \ldots, r_n$}:

$$\begin{cases} r_i \le r_{2i} \\ r_i \le r_{2i+1} \end{cases}$$ (小顶堆)　　或　　$$\begin{cases} r_i \ge r_{2i} \\ r_i \ge r_{2i+1} \end{cases}$$ (大顶堆)

# 讨 论

思考：倒置小顶堆是否一定是大顶堆？

{12, 36, 27, 65, 40, 34, 98, 81, 73, 55, 49}　　是小顶堆

{12, 36, 27, 65, 40, 14, 98, 81, 73, 55, 49}　　不是堆

# 4.2.4 堆排序

## 堆排序

- 思想

(1) 以初始关键字序列，建立堆；

(2) 输出堆顶最小元素；

(3) 调整余下的元素，使其成为一个新堆；

(4) 重复(2),(3) 次，直到n个元素输出，得到 一个有序序列.

关键要解决(1)和(3)，即如何由一个无序序列建成一个堆?

如何调整余下的元素成为一个新堆?

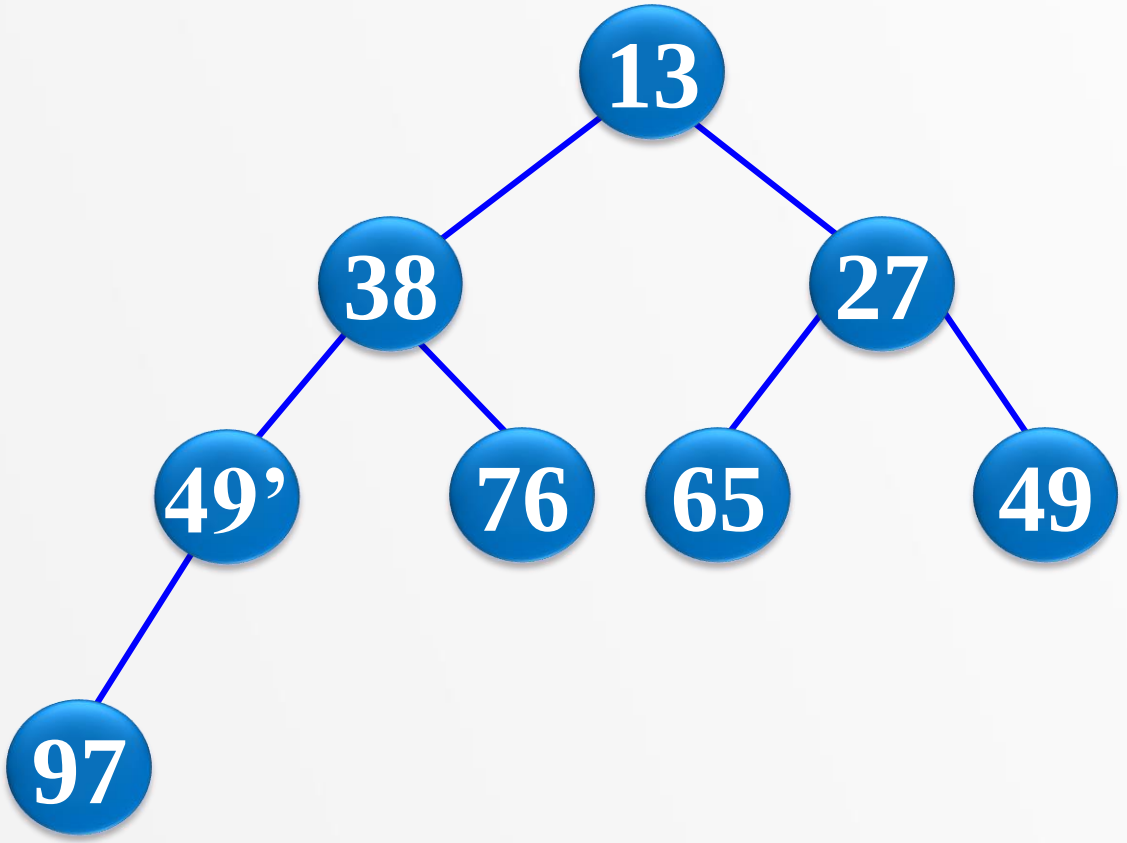
# 4.2.4 堆排序

## 堆排序

- 调整

输出: 13 27 38

例如：设有关键字{13,38,27,49’ ,76,65,49,97}，按初始次序构成一棵完全二叉树，形成一个堆。



```
void shift(DataType r[], int k, int m){
 /*假设r[k+1, ..., m]满足小顶堆的性质，本算法调整
   r[k]使得整个序列r[k, ..., m]满足小顶堆的性质*/
 i=k; j=2*i; x=r[k].key; finished=false;
 t=r[k];  /*暂存根的数据*/
 while(j<=m & !finished){
      if (j<m & r[j].key>r[j+1].key) j=j+1;    /* j+1<=m */
      /* 若存在右子树，且右子树根的关键字小，沿右分支筛选*/
      if(x<=r[j].key) finished=true; /*筛选完毕*/
      else {r[i]=r[j];i=j; j=2*i;}
 }
 r[i]=t;
}
```

# 讨 论

为什么堆适合采用顺序存储结构?

# 4.2.4 堆排序

## 初始序列建堆

例：{49，38，65，97，76，13，27，49’ }

堆排序