第五章 排序

part 2: 选择排序 与 交换排序

张史梁
slzhang.jdl@pku.edu.cn

内容提要

- 排序的基本概念
- 插入排序
- 选择排序
- 交换排序
- 分配排序
- 归并排序

排序的基本概念

- 正序与逆序
  - “正序”序列：待排序序列正好符合排序要求
  - “逆序”序列：把待排序序列逆转过来，正好符合排序要求
  - 譬如，需要得到一个非递减序列？

- 正序：

11	19	23	55	80	97

- 逆序：

97	80	55	23	19	11

排序的基本概念

- 排序的稳定与不稳定：
  - 在待排序的文件中，若存在多个排序码相同的记录，
  - 经过排序后记录的相对次序保持不变，则这种排序方法称为是“稳定的”；
  - 否则，是“不稳定的”(只需列举出一组关键字说明不稳定即可)
  - 譬如：对下列数据进行排序，得到一个递增序列

23	19’	55	97	19	80

排序过程中出现

19’	19	23	97	55	80

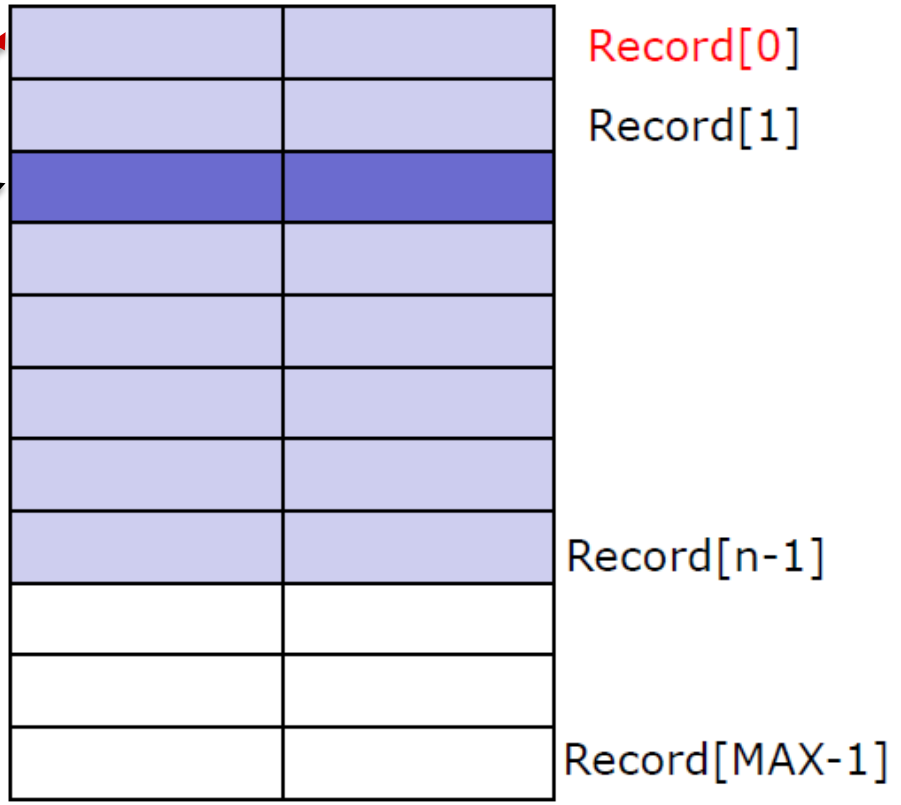
排序的基本操作

- 比较两个排序码的大小
- 将一个记录从一个位置移动到另一个位置

本章假设-记录的数据结构

```
typedef int KeyType;
typedef int DataType;
typedef struct
{
    KeyType key; /* 排序码字段 */
    DataType info; /* 其他字段*/
}RecordNode;

typedef struct
{
    RecordNode record[MAX];
    int n;  // 记录个数，n<MAX
}SortObject;
```



选择排序

- 思想：每趟从待排序的记录序列中选择关键字最小的记录放置到已排序表的最前位置，直到全部排完。

- 关键问题：在剩余的待排序记录序列中找到最小关键码记录。（找最小值问题）

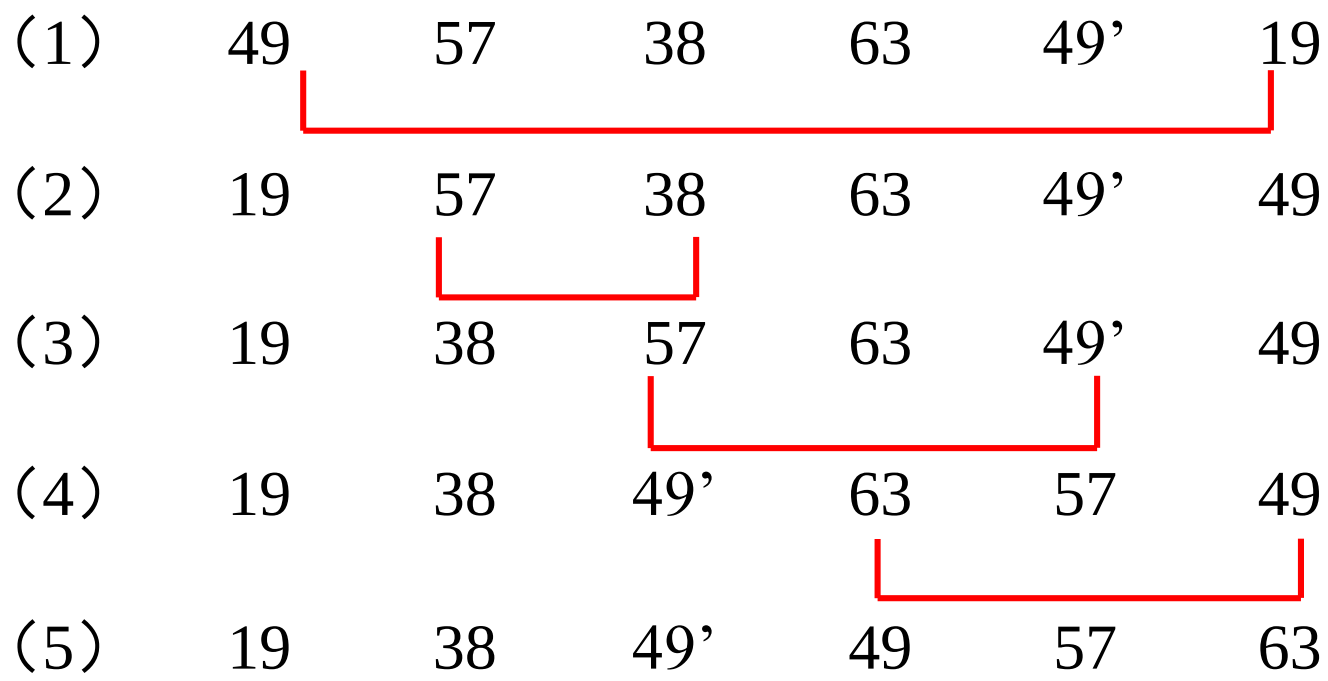
- 方法：
  - 直接选择排序
  - 堆排序

直接选择排序

- 基本思想：
  - 首先在所有记录中选出排序码最小的记录，与第一个记录交换
  - 然后在其余的记录中再选出排序码最小的记录与第二个记录交换

    。 。 。
  - 以此类推，直到所有记录排好序

直接选择排序 – 示例



注意，n个元素，只需要选择n-1次。另外，正序和逆序影响比较次数吗？

直接选择排序算法

```
①  void selectSort(SortObject * pvector) // 求递增序列
②  {
③     int i, j, k;   RecordNode temp;
④     for( i = 0; i < pvector->n-1; i++ )   /* 做n-1趟选择排序*/
⑤     {
⑥        k=i;
⑦        for(j=i+1; j<pvector->n; j++)    /*找出排序码最小的Rk */
⑧          if(pvector->record[j].key<pvector->record[k].key)   k=j;
⑨        if(k!=i)                          /*记录Rk 与Ri 互换*/
⑩        {
⑪         temp=pvector->record[i];
⑫         pvector->record [i]= pvector->record [k];
⑬         pvector->record [k]=temp;
⑭        }
⑮     }
⑯  }
```

教学网提供代码：selectsort

直接选择排序性能分析

- 选择排序的比较次数与记录的初始状态无关。
  - 第i趟排序：从第i个记录开始，顺序比较选择最小关键码记录需要n-i次比较。
  - 总的比较次数：$\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}$
  - 最少移动次数：$M_{min}=0$ （初始为正序时）
  - 最多移动次数：$M_{max}=3(n-1)$ (初始为逆序，每趟1次交换，需3次移动完成）
- 时间复杂度：$T(n)=O(n^2)$
- 辅助空间1个记录单位：Temp
- 稳定性：不稳定的排序
  - 示例中，第一步的选择将最小元素13与第一个元素49进行了交换时，已经改变了49与49’的相对顺序

选择排序

- 思想：每趟从待排序的记录序列中选择关键字最小的记录放置到已排序表的最前位置，直到全部排完。

- 关键问题：在剩余的待排序记录序列中找到最小关键码记录。

- 方法：
  - 直接选择排序
  - 堆排序

堆排序

- 动机：在后面的选择过程中利用前面选择时的比较结果以减少比较次数。

- 由1991年计算机先驱奖获得者、斯坦福大学计算机科学系教授Robert W．Floyd 和J．Williams在1964年共同发明

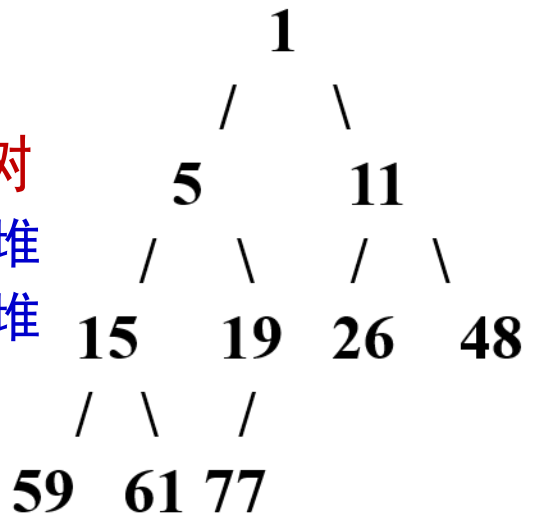
- 堆排序加速了查找最小（最大）元素的过程

堆排序

- 堆的定义：n个排序码序列$K=\{k_0,k_1,k_2,\ldots,k_{n-1}\}$当且仅当满足如下条件时，称之为堆。

$$k_i \le k_{2i+1} \quad k_i \le k_{2i+2} \qquad \text{或} \qquad k_i \ge k_{2i+1} \quad k_i \ge k_{2i+2} \quad (i=0, 1, 2, ..., n/2-1)$$

0	1	2	3	4	5	6	7	8	9
1	5	11	15	19	26	48	59	61	77



- 从定义看出：堆实质上是一棵完全二叉树
  - 如果堆中根结点的排序码最小，则称为小根堆
  - 如果堆中根结点的排序码最大，则称为大根堆
  - 若设二叉树的深度为h，除第 h 层外，其它各层 (1～h-1) 的结点数都达到最大个数
  - 第 h 层所有的结点都连续集中在最左边，这就是完全二叉树。

重要特性：堆中含堆，所以根节点一定是最大/最小值

堆排序的主要思想（从小到大排）

- 基本思想：

① 把原始序列构造成一个堆，使得n个元素的最大值处于序列的第一个位置；

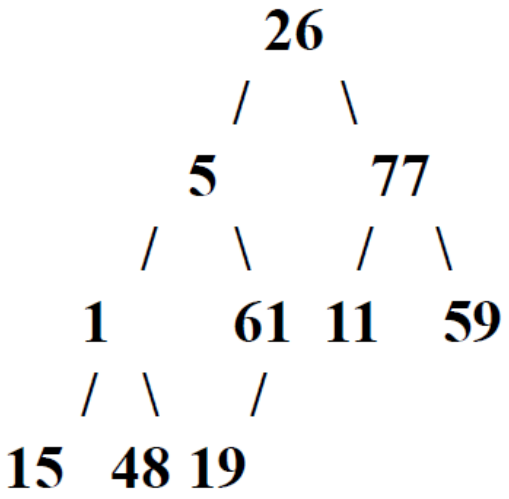
② 然后交换序列第1个元素(最大值元素)与第n个元素；

③ 再把序列的前n-1个元素组成的子序列构成一个新堆，得到第二大元素，把序列的第一个元素与第n-1个元素交换。此后再把序列的前n-2个元素构成一个新堆……

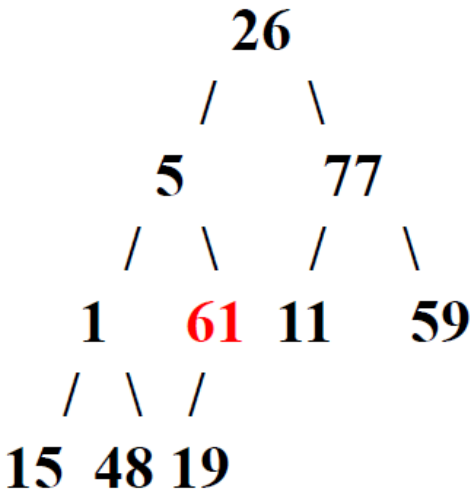
④ 反复此操作，最终整个序列成为有序序列。

堆排序 – 示例构造大根堆

初始序列为 26，5，77，1，61，11，59，15，48，19

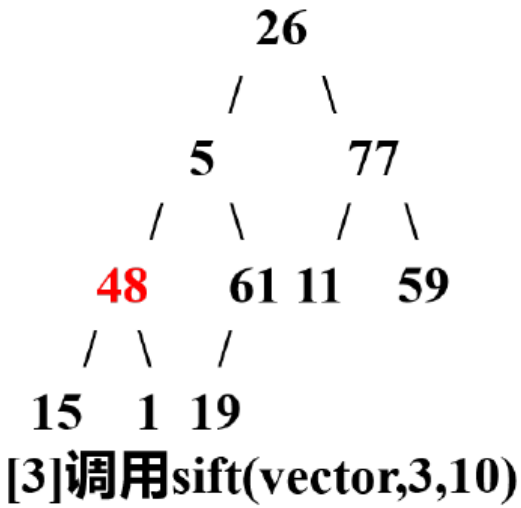


[1] 初始完全二叉树

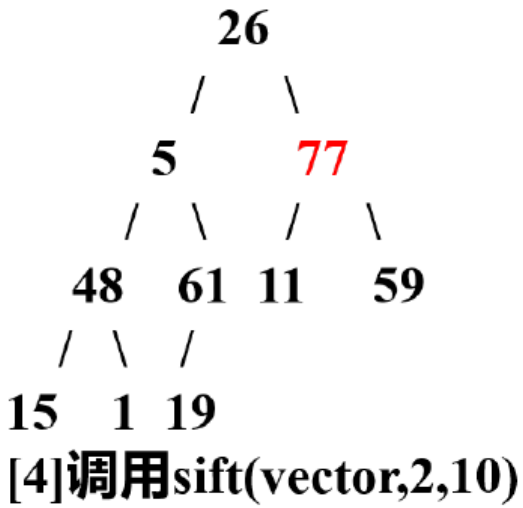


[2] i=⌊n/2⌋-1=4 , 调用 sift(vector,4,10)

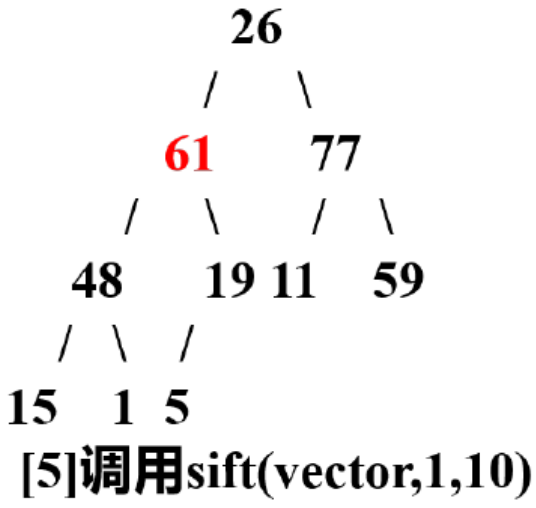
堆排序 – 示例构造大根堆



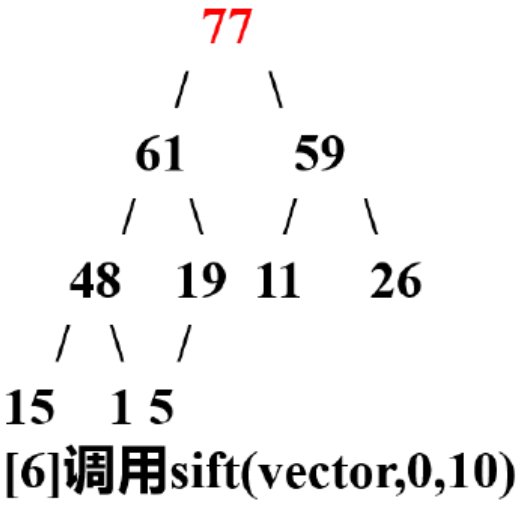
[3]调用sift(vector,3,10)



[4]调用sift(vector,2,10)

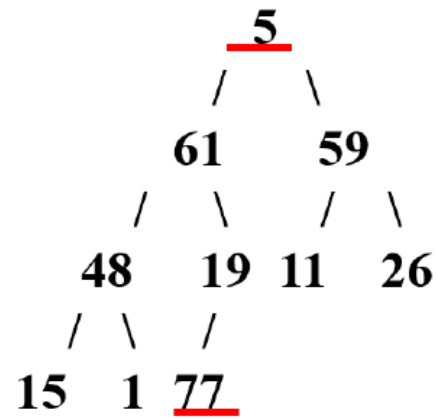


[5]调用sift(vector,1,10)

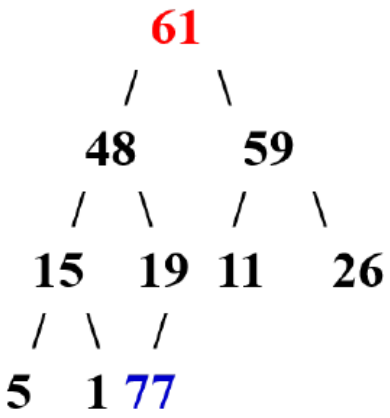


[6]调用sift(vector,0,10)

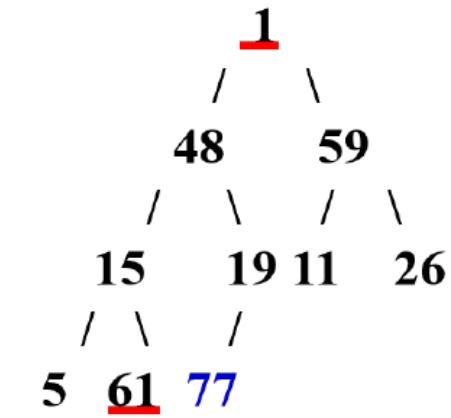
堆排序 – 示例构造大根堆



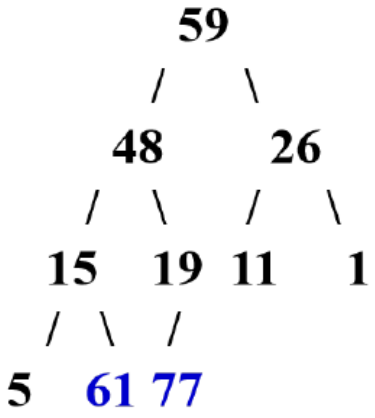
[7] 77与5互换



[8] 重建堆

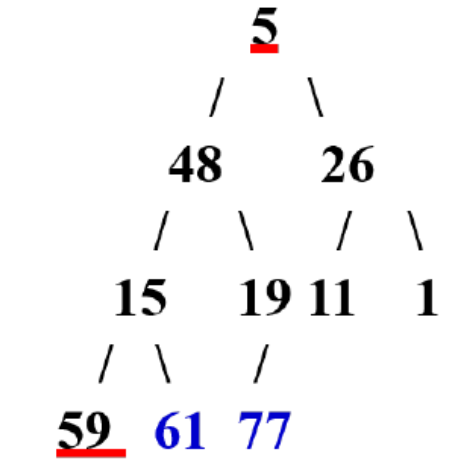


[9] 61与1互换

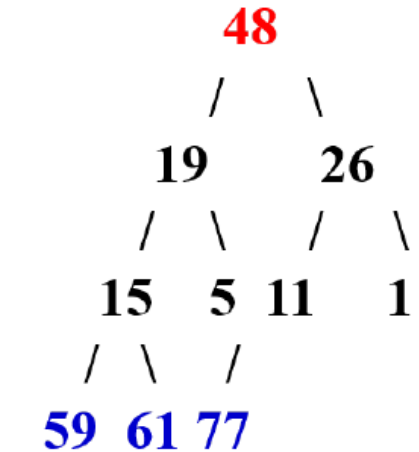


[10] 重建堆

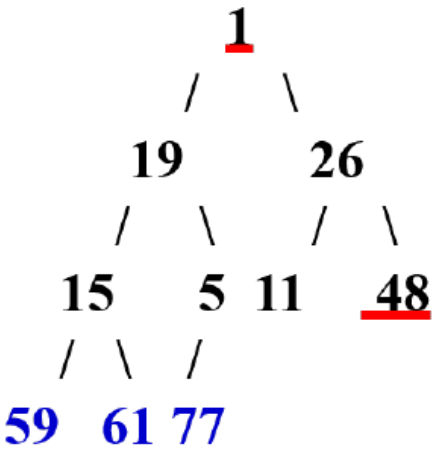
堆排序 – 示例构造大根堆



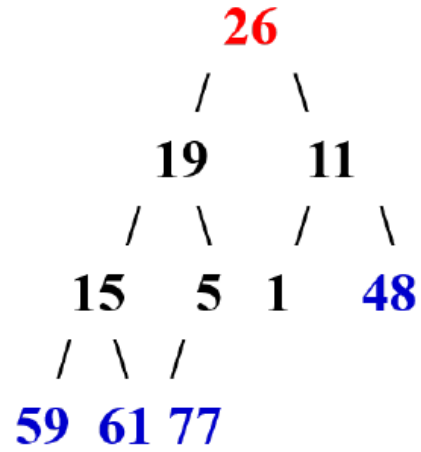
[11] 59与5互换



[12] 重建堆

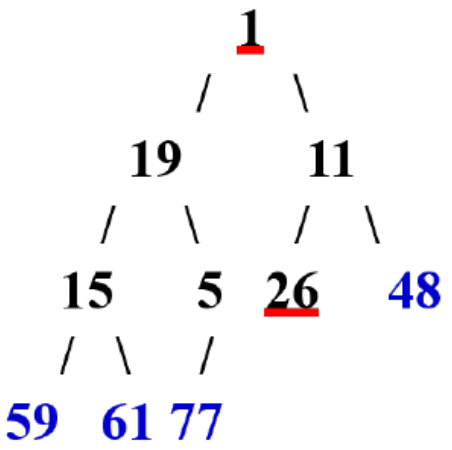


[13] 48与1互换

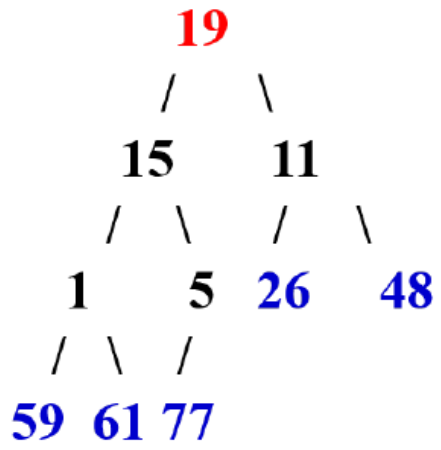


[14] 重建堆

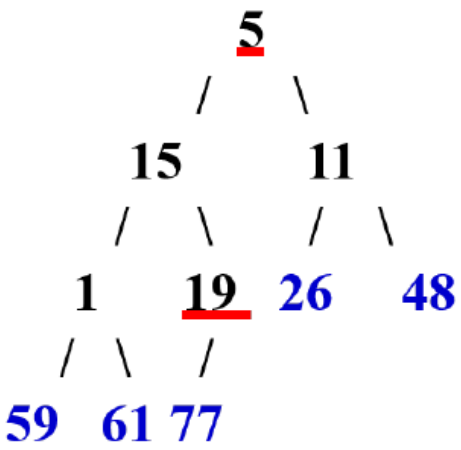
堆排序 – 示例构造大根堆



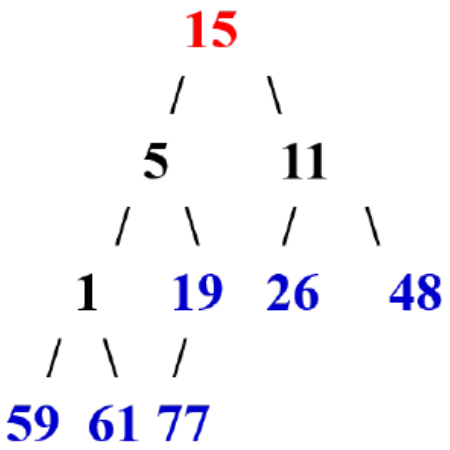
[15] 26与1互换



[16] 重建堆

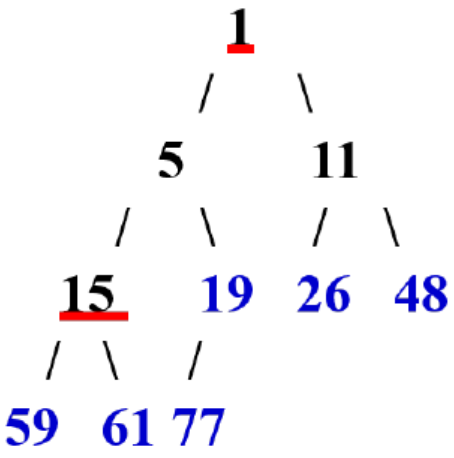


[17] 19与5互换

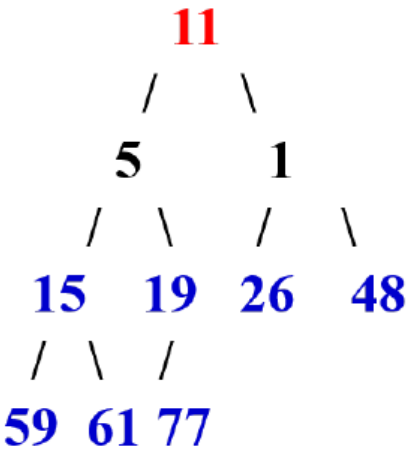


[18] 重建堆

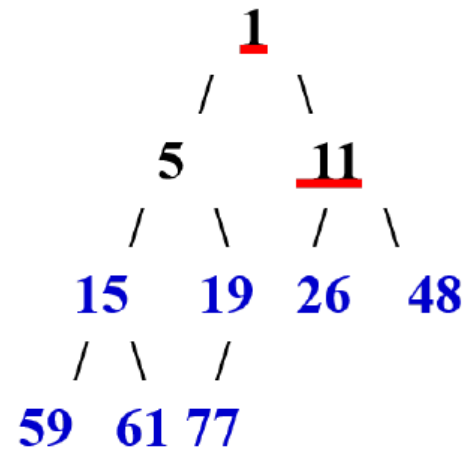
堆排序 – 示例构造大根堆



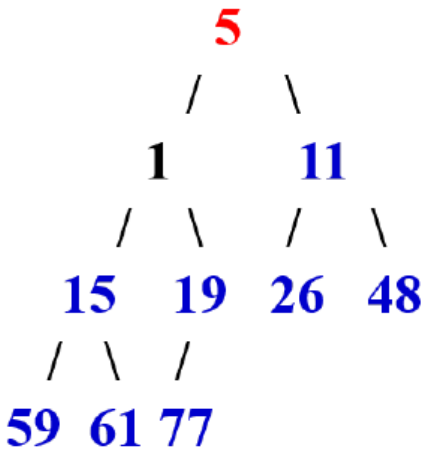
[19] 15与1互换



[20] 重建堆

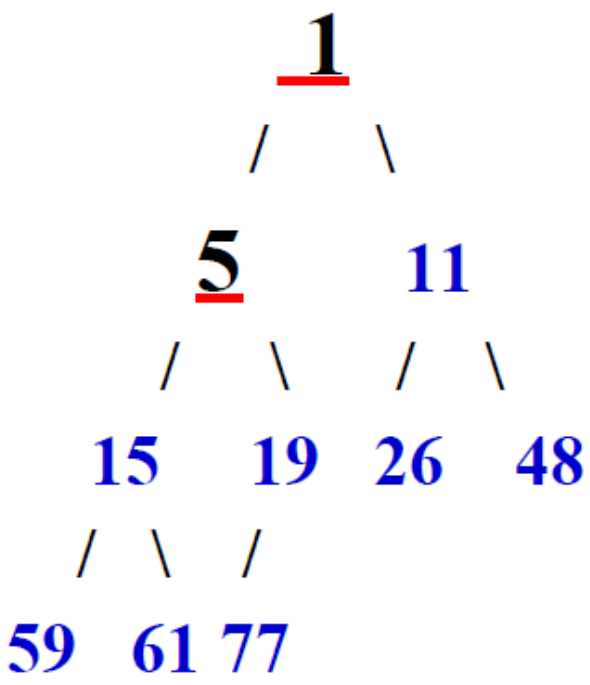


[21] 11与1互换



[22] 重建堆

堆排序 – 示例构造大根堆

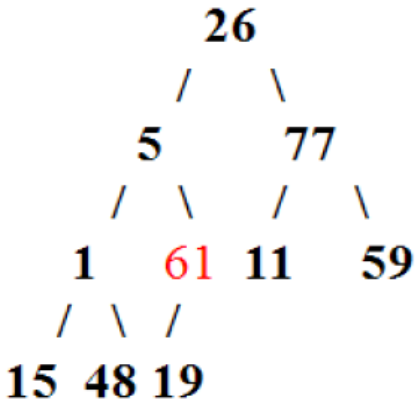


[23] 5与1互换(排序完成）

关键问题 -1

- 如何将原始序列构成初始堆？
  - 初始完全二叉树中，⌊n/2⌋，⌊n/2⌋+1，…，n-1为叶子，以其为根的子树必然为堆。
  - 因此，初始堆建立时，只需要将所有非终端（非叶子）结点为根的子树调整为堆。

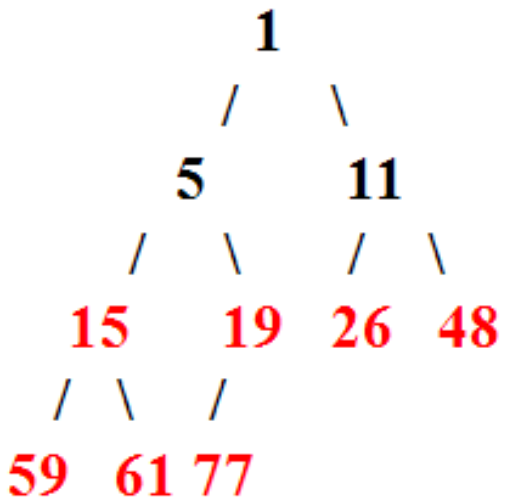
0, 1, …, ⌊n/2⌋-1



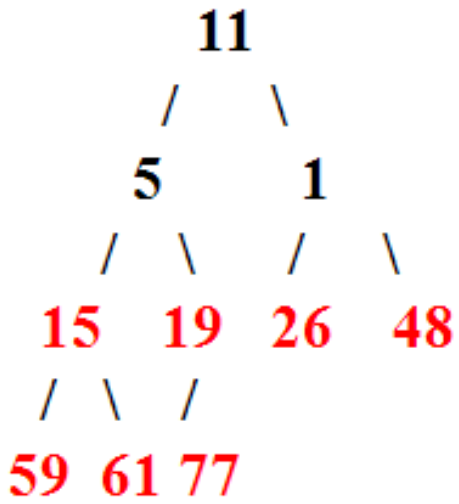
[2] i=⌊n/2⌋-1=4 , 调用 sift(vector,4,10)

关键问题 -2

- 初始建堆以后，把序列的第一个元素与第n个元素交换，丢掉最大值后如何构造新堆？



[19] 15与1互换



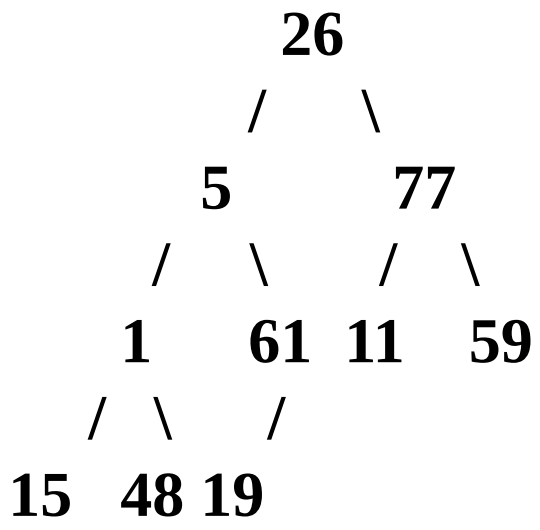
[20] 重建堆

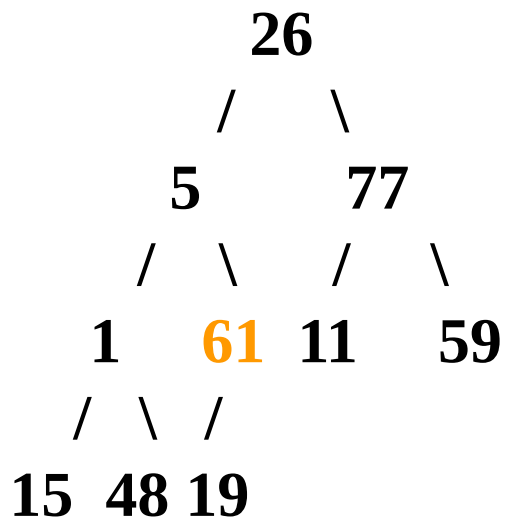
关键问题 -2

- 可采用“筛选法”建堆
  - 为以R_i为根的完全二叉树建堆，这时R_i 的左、右子树都是堆，可以把R_i与其左、右子树根结点R_{2i+1}、R_{2i+2}中最大者交换位置。
  - 若交换位置后破坏了子树的堆特性，则再对这棵子树重复交换过程（重复筛选法），直到以R_i为根结点的子树成为堆。

示例

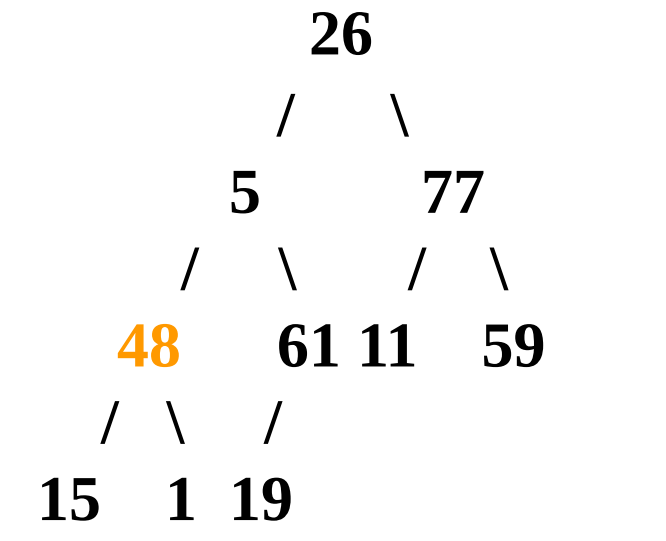
初始序列为 26，5，77，1，61，11，59，15，48，19



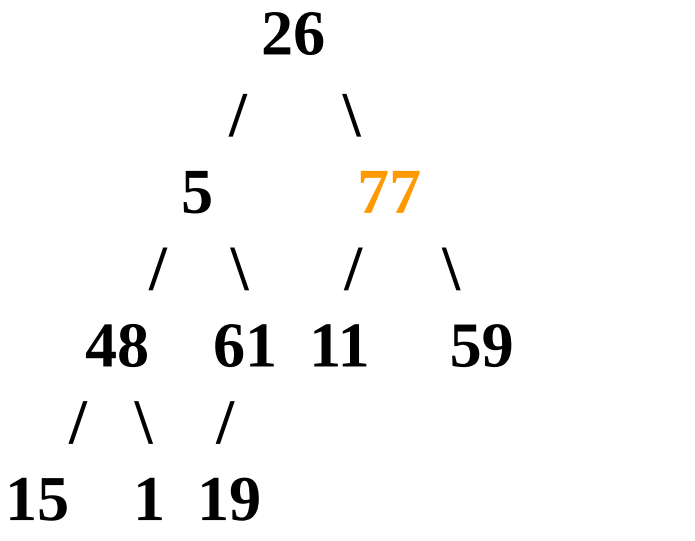


[1] 初始完全二叉树　　**[2] i=⌊n/2⌋-1=4 , 调用sift(vector, 4, 10)**

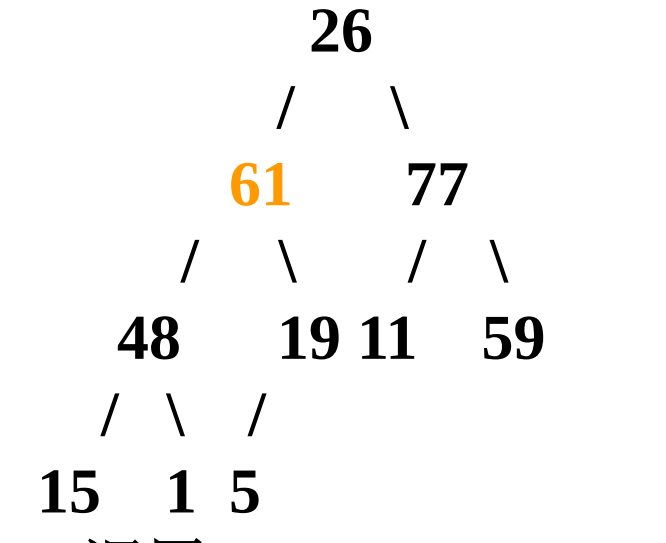
```
for(i=n/2-1;i>0;i--)     sift(pvector,i,n);     /* 建立初始堆*/
```



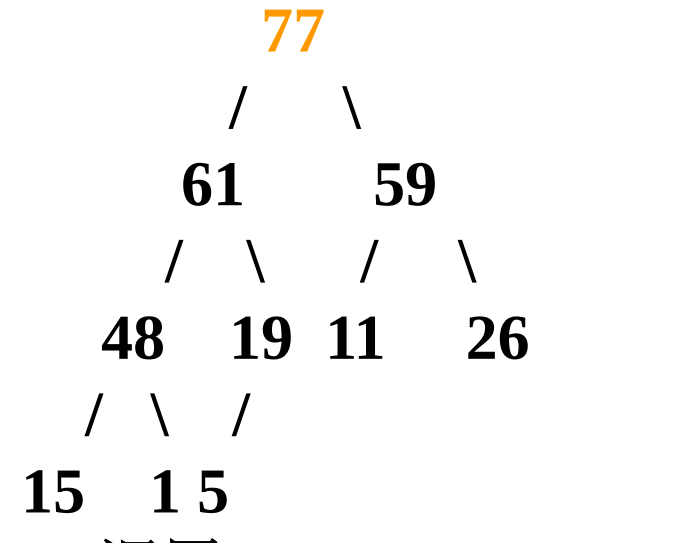
[3]调用sift(vector, 3, 10)



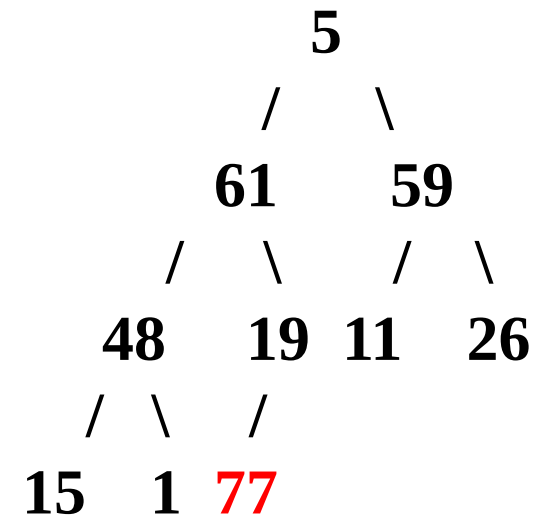
[4]调用sift(vector, 2, 10)



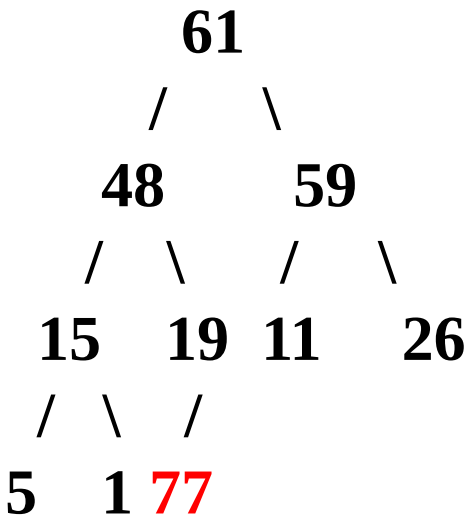
[5]调用sift(vector, 1, 10)



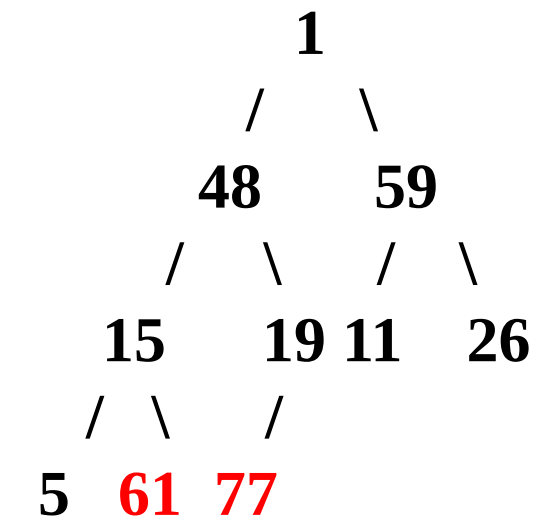
[6]调用sift(vector, 0, 10)



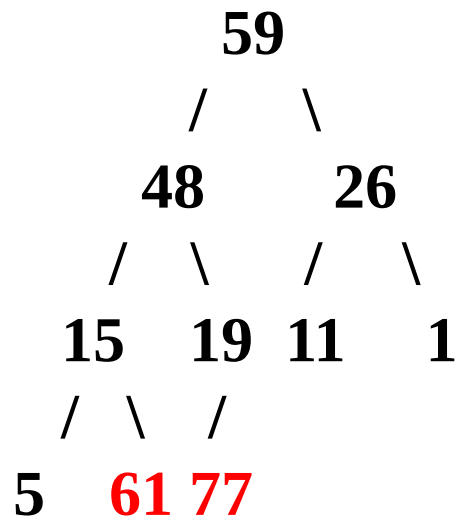
[7] 77与5互换



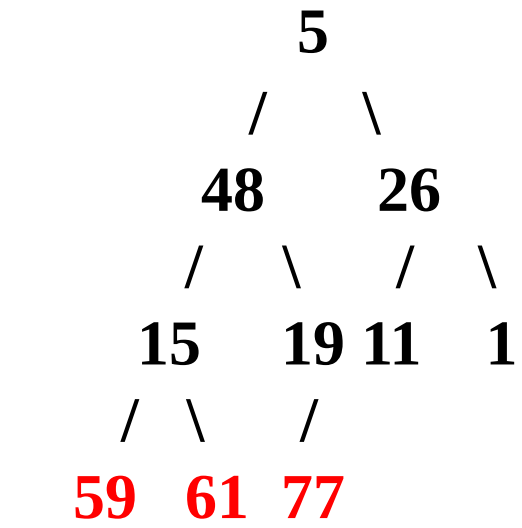
[8] 重建堆



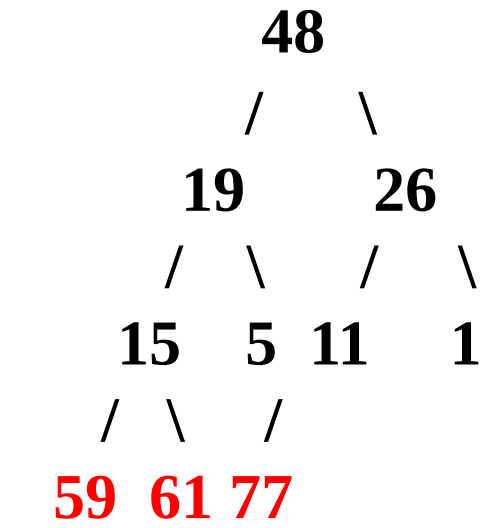
[9] 61与1互换



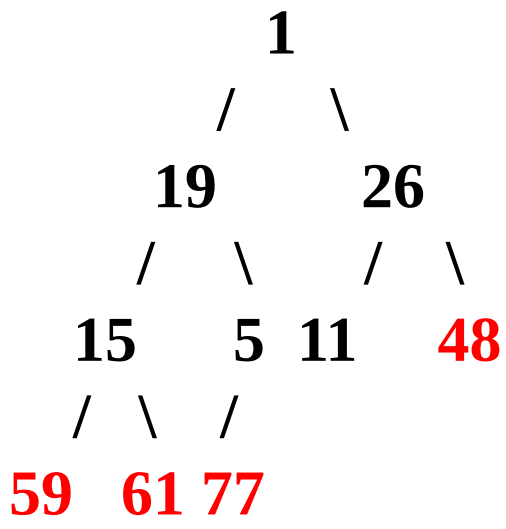
[10] 重建堆



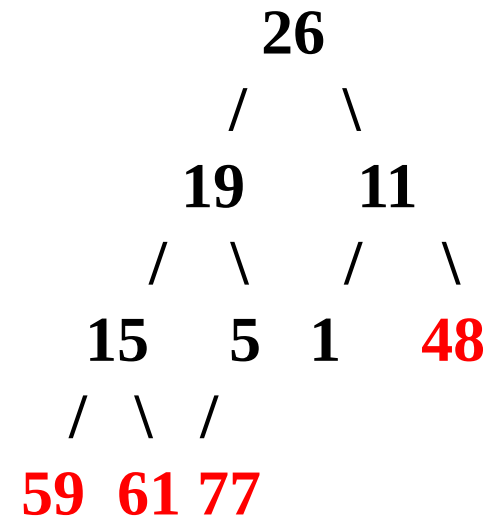
[11] 59与5互换



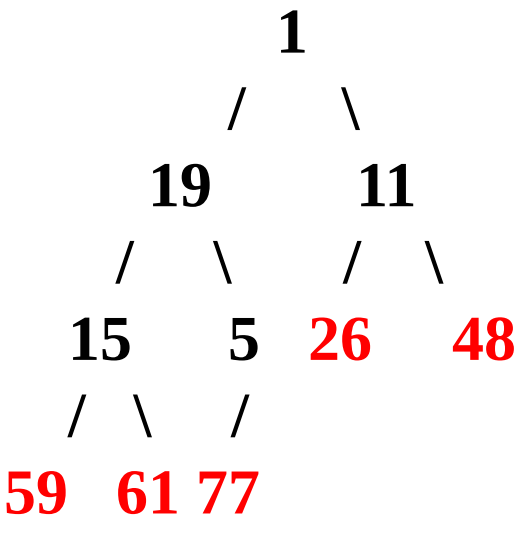
[12] 重建堆



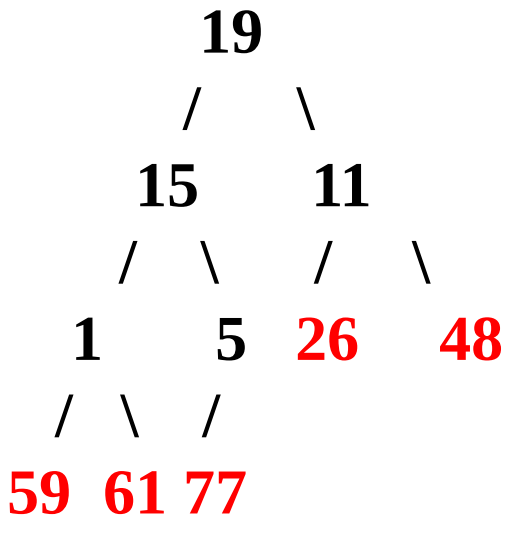
[13] 48与1互换



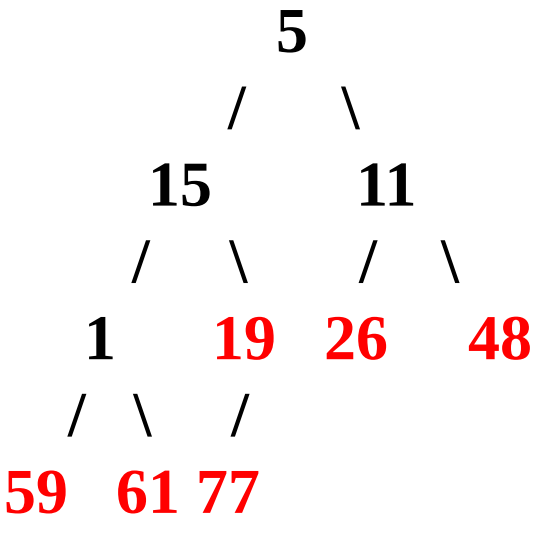
[14] 重建堆



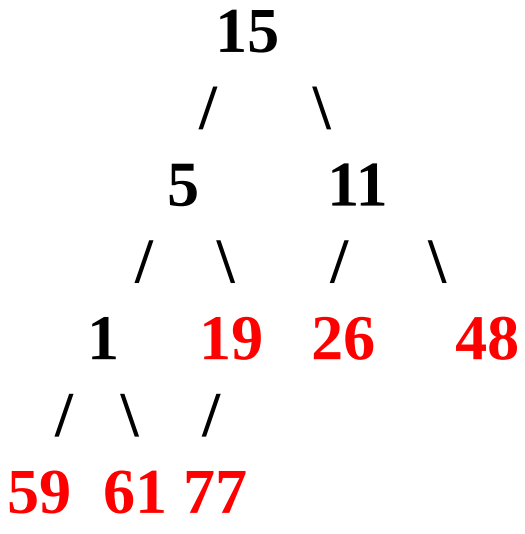
[15] 26与1互换



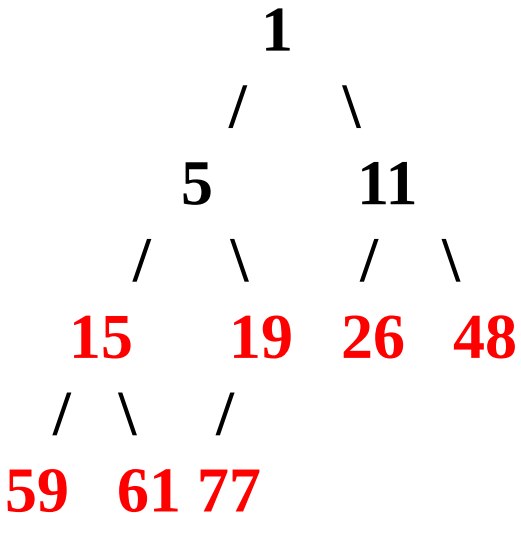
[16] 重建堆



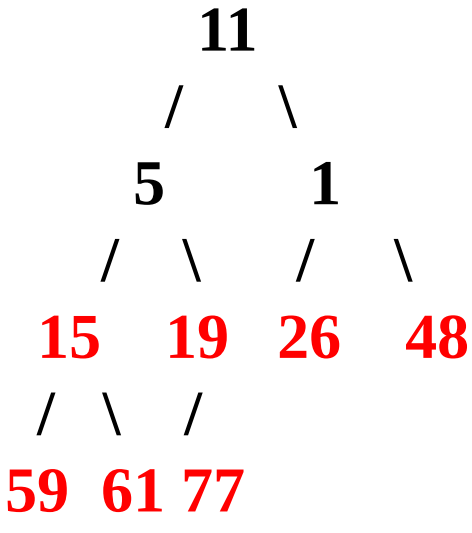
[17] 19与5互换



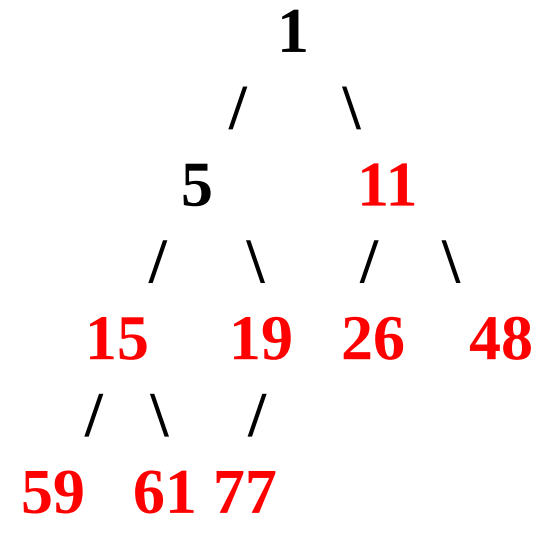
[18] 重建堆



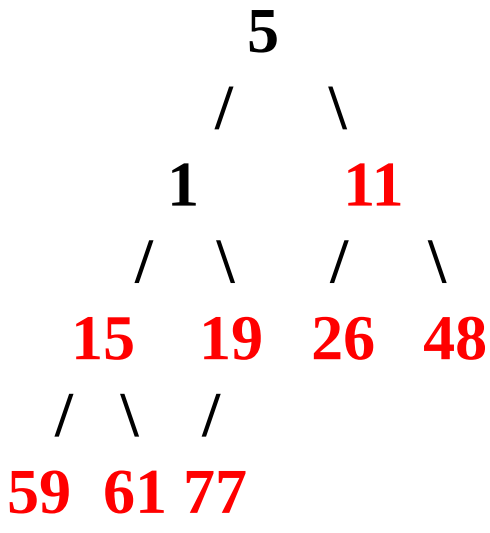
[19] 15与1互换



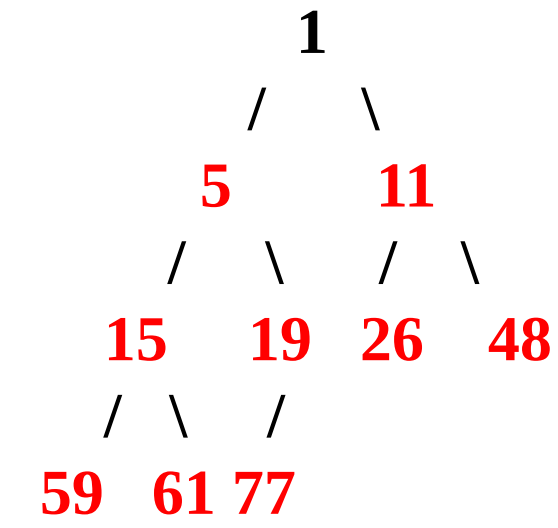
[20] 重建堆



[21] 11与1互换



[22] 重建堆



[23] 5与1互换（排序完成）

1，5，11，15，19，26，48，59，61，77

堆排序算法

```
①  void heapSort( SortObject *pvector)  /*堆排序*/
②  {
③      int i,n;
④      RecordNode temp;
⑤      n=pvector->n;
⑥      for(i=n/2-1;i>0;i--)     sift(pvector, i, n);     /* 建立初始堆*/
⑦      for (i=n-1;i>0;i--)    /*进行n-1趟堆排序*/
⑧      {
⑨          temp=pvector->record[0];     /* 当前堆顶和最后记录互换*/
⑩          pvector->record[0]=pvector->record[i];
⑪          pvector->record[i]=temp;
⑫          sift(pvector,0,i);                    /* 从R0到Ri-1重建堆  */
⑬      }
⑭  }
```

筛选算法

```
①  #define leftchild(i) 2*(i)+1                    //左子节点下标计算
②  void sift(SortObject *p, int i, int n)
③  {
④     int child;  RecordNode temp;
⑤     temp=p->record[i];                           //存储Ri
⑥     child=leftchild(i);    /*Rchild是R0的左子女*/
⑦     while(child <n)
⑧     {
⑨        if((child<n-1)&&(p->record[child].key<p->record[child+1].key))
⑩           child++;  /* child 指向Ri的值较大的子结点*/
⑪        if( temp.key < p->record[child].key)
⑫        {
⑬           p->record[i]=p->record[child];
⑭           /*将Rchild换到父结点位置，进入下一层继续调整*/
⑮           i=child;  child=leftchild(i);     //对交换以后的子节点使用sift算法
⑯        }
⑰        else break;  /*调整结束*/
⑱     }
⑲     p->record[i]=temp;  /*将记录Ri放入正确位置*/
⑳  }
```

时间效率评价

- 分析
  - 建初始堆比较次数C_1：$O(n)$
  - 重新建堆比较次数C_2：$O(n\log_2 n)$
  - 总比较次数=C_1+C_2
  - 移动次数小于比较次数，因此，
- 时间复杂度：$O(n\log_2 n)$
- 空间复杂度：$O(1)$

- 与简单排序算法的$O(n^2)$相比，有根本性改善，适用于n值较大的情况。
- 算法稳定性：不稳定

数据初始分布对效率没有太大影响

内容提要

- 排序的基本概念
- 插入排序
- 选择排序
- 交换排序
- 分配排序
- 归并排序

交换排序

- 交换排序的基本方法
  - 两两比较待排序记录的排序码，交换不满足顺序要求的偶对，直到全部满足为止
- 交换排序的分类
  - 起泡排序
  - 快速排序

起泡排序

- 基本思想：
  - 一次起泡：先将序列中的第一个记录R_0与第二个记录R_1比较，若前者大于后者，则两个记录交换位置，否则不交换
  - 然后对新的第二个记录R_1与第三个记录R_2作同样的处理
  - 依次类推，直到处理完第n-1个和第n个记录 (R_{n-2}，R_{n-1})
  - 从(R_0，R_1)到(R_{n-2}，R_{n-1})的n-1次比较和交换过程称为一次起泡
  - 经过这次起泡，n个记录中最大者被安置在第n个位置上
  - 以此类推，再对前n-1个记录进行同样处理 …

起泡排序 – 示例【向后起泡】

初始序列为 49, 38, 65, 97, 76, 13, 27, 49’，用起泡排序法排序

第一趟起泡：

38　　49　　65　　76　　13　　27　　49’　　[97]

第二趟起泡：

38　　49　　65　　13　　27　　49’　　[76　　97]

第三趟起泡：

38　　49　　13　　27　　49’　　[65　　76　　97]

起泡排序 – 示例【向后起泡】

第四趟起泡：

38 13 27 49 [49’ 65 76 97]

第五趟起泡：

13 27 38 [49 49’ 65 76 97]

第六趟起泡：

13 27 [38 49 49’ 65 76 97]

排序结果为：

13 27 38 49 49’ 65 76 97

起泡排序 – 示例【向前起泡】

初始序列为 49, 38, 65, 97, 76, 13, 27, 49’，用起泡排序法排序

第一趟起泡：【13】49, 38, 65, 97, 76, 27, 49’

第二趟起泡：【13 27】49, 38, 65, 97, 76, 49’

第三趟起泡：【13 27 38】49, 49’, 65, 97, 76

第四趟起泡：【13 27 38 49】49’, 65, 76, 97

第五趟起泡：【13 27 38 49 49’】65, 76, 97

第六趟起泡：【13 27 38 49 49’ 65】76, 97

排序结果为：【13 27 38 49 49’ 65 76】97

起泡排序方法改进

- 改进：
  - 可以设置一个标志(noswap)表示本次起泡是否有记录交换，如果没有交换则表示整个排序过程完成

起泡排序算法 【向后起泡】

```
void bubbleSort(SortObject * pvector)
{
   int i, j, noswap;
   RecordNode temp;
   for(i=0; i<pvector->n-1; i++)      /* 做n-1次起泡 */
   {
      noswap=TRUE;
      for(j=0; j<pvector->n-i-1; j++)   /*置交换标志*/
         if(pvector->record[j+1].key<pvector->record[j].key)
                                        /*从前向后扫描*/
         {
            temp=pvector->record[j];
            pvector->record[j]=pvector->record[j+1];
            pvector->record[j+1]=temp;
            noswap=FALSE;
         }
      if(noswap) break;  /*本趟起泡未发生记录交换，算法结束*/
   }
}
```

起泡排序的算法评价

- 若文件初状为正序，则一趟起泡就可完成排序，排序码的比较次数为n-1，且没有记录移动，时间复杂度是O(n)
- 若文件初态为逆序，则需要n-1趟起泡，每趟进行n-i次排序码的比较，且每次比较都移动三次，比较和移动次数均达到最大值：

$$C_{\max} = \sum_{i=1}^{n-1}(n-i) = n(n-1)/2 = O(n^2)$$

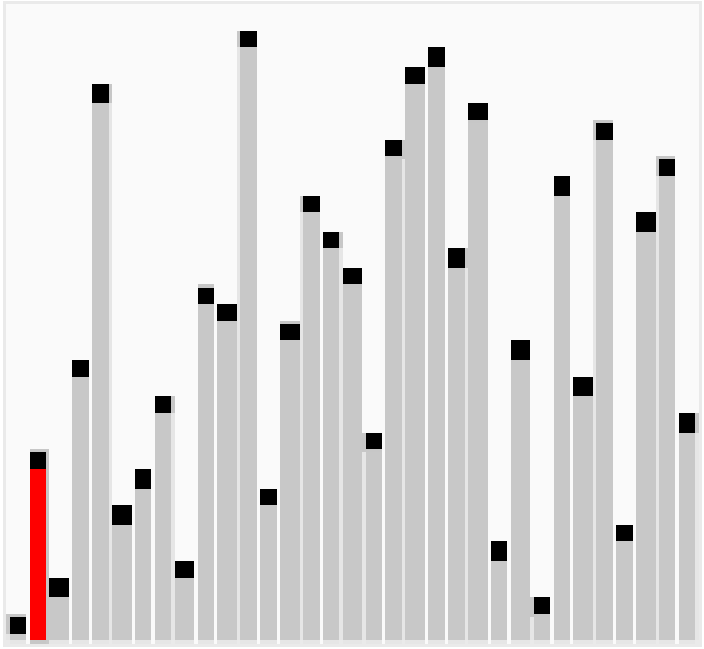
$$M_{\max} = \sum_{i=1}^{n-1}3(n-i) = 3n(n-1)/2 = O(n^2)$$

起泡排序的算法评价(续)

- 起泡排序最坏时间复杂度为$O(n^2)$
- 起泡排序平均时间复杂度为$O(n^2)$
- 起泡排序算法中增加一个辅助空间temp，辅助空间为$S(n)=O(1)$
- 起泡排序是稳定的

起泡排序方法改进

- 改进：
  - 可以设置一个标志(noswap)表示本次起泡是否有记录交换，如果没有交换则表示整个排序过程完成
  - 双向起泡排序



(2,3,4,5,1)：
单向起泡：4趟
双向起泡：2趟

交换排序

- 起泡排序
- 快速排序

快速排序算法是20世纪十大算法之一，它于1962年由Tony Hoare提出。

17. 计算的美丽 – 1980 年图灵奖获得者 Tony Hoare

C. Antony R. Ho

Charles Antony Richard Hoare(Tony Hoare和CAR Hoare)出生于 1934 年 1 月 11 日于斯里兰卡。其父母为英国人。Hoare于 1956 年从牛津大学(http://www.ox.ac.uk/)获得其学士学位。后来Hoare前往原苏联并在莫斯科州立大学学习自然语言的计算机转换。1960 年，Hoare回到英国并工作于Elliott Brothers公司。在Elliott Brothers，Hoare实现了ALGOL 60 编译器。1968 年，Hoare获得了University of Belfast(http://www.qub.ac.uk/)的教授职务。1977 年，Hoare回到牛津大学担任计算机科学程序语言研究小组的教授。除了其在牛津大学的教职，Hoare也在微软Microsoft Inc.在英国的研究所出任研究员的职位，可参见：http://research.microsoft.com/users/thoare/

在 2000 年，Hoare 由于其在计算机科学和教育方面的杰出贡献被英国皇家授予爵士爵位。

图灵奖获得时间：

1980 年。 第十五位图灵奖(1980

图灵奖引用(Turing Award Citation)：

For his fundamental contributions to the definition and design of programming languages.

笔者注：

Hoare 对程序设计语言的主要贡献为：Hoare Logic, 快速排序算法(Quicksort)和 CSP(Communication Sequential Processes)

快速排序

- 快速排序是对起泡排序的改进
  - 起泡排序，是在相邻两个记录间比较和交换，每次交换只能上移或下移一个位置，导致总的比较与移动次数增多
- 快速排序又称分区交换排序
  - 设待排序的n个记录{$R_0, R_1, \ldots, R_{n-1}$}，选取第一个记录$R_0$为标准，寻找$R_0$的最终位置（一趟快速排序）：
    - [R[0], R[i-1]] 存放的为小于R_0的记录
    - [R[i+1], R[n-1]] 存放的为大于R_0的记录
    - R[i] 为R_0的最终位置
  - 快速排序方法记录间比较次数较少，因此速度较快，被认为是较好的排序方法

快速排序基本思想

- 在待排序的n个记录中任取一个记录(通常取第一个记录)，
  - 把所有小于该记录的记录移到其左边，
  - 把所有大于该记录的记录移到其右边，
  - 所选记录正好处在其应在的位置，
  - 且把原有序列划分成两个子序列
- 然后，对两个子序列分别重复上述过程，直到所有记录都排好序
- 把当前参加排序的记录按第一个记录分成前后两个部分的过程称为一趟快速排序

一趟快速排序

- 设置变量i= 0，变量j= n-1；
- 保存记录temp=R_0，R[0]为空出的位置（空位在前一区）；
- 令j向前扫描，寻找小于R_0的记录，找到小于R_0的记录R[j]，将记录R[j]移到当前空位中。这时R[j]为新空位（空位在后一区）；
- i自i+1起向后扫描，寻找大于R_0的记录，找到大于R_0的记录R[i]，将记录R[i]移到当前空位中，空位又到了前一区（R[i]的位置）；
- 如此交替改变扫描方向，从两端向中间靠拢，直到i=j，这时i所指的位置为R_0的最终位置

一趟快速排序 – 示例

初始序列为49, 38, 65, 97, 76, 13, 27, 49’，用快速排序法排序

(1)一次分区过程如下：

j向左扫描

[__ 38 65 97 76 13 27 49’] 49

i↑ j↑

第一次交换后

[27 38 65 97 76 13 _ 49’]

i↑ j↑

i向右扫描

[27 38 65 97 76 13 _ 49’]

i↑ j↑

一趟快速排序 – 示例

第二次交换后

[27 38 _ 97 76 13 65 49’]

i↑ j↑

j向左扫描，位置不变，第三次交换后

[27 38 13 97 76 _ 65 49’]

i↑ j↑

i向右扫描，位置不变，第四次交换后

[27 38 13 _ 76 97 65 49’]

i↑ j↑

j向左扫描

[27 38 13 49 76 97 65 49’]

i↑j↑

快速排序的基本思想

- 首先，在一趟快速排序过程中，在待排序的n个记录$\{R_0, R_1, \ldots, R_{n-1}\}$中任选一个记录为标准（通常为$R_0$），把这n个记录按“标准”分成前后两个部分：
  - 把所有小于该记录的记录移到其左边，
  - 把所有大于该记录的记录移到其右边，
  - 所选记录正好处在其应在的位置，且把原有序列划分成两个子序列
- 然后，对两个子序列分别重复上述过程，直到所有记录都排好序

快速排序 – 示例

序列{49，38，65，97，76，13，27，49’}在快速排序过程中各趟排序后的状态

[27	38	13]	49	[76	97	65	49’]
[13]	27	[38]	49	[76	97	65	49’]
13	27	[38]	49	[76	97	65	49’]
13	27	38	49	[76	97	65	49’]
13	27	38	49	[49’	65]	76	[97]
13	27	38	49	49’	[65]	76	[97]
13	27	38	49	49’	65	76	[97]
13	27	38	49	49’	65	76	97

快速排序算法

```
void quickSort(SortObject * pvector, int l, int r)
{
   int i, j;
   RecordNode temp;
   if(l>=r) return;      /*只有一个记录或无记录，则无需排序*/
   i=l; j=r; temp=pvector->record[i];   // temp保存R0
   while(i!=j)           /*寻找R0的最终位置*/
   {
      while( (pvector->record[j].key>=temp.key) && (i<j) )
         j--;  /*从右到左扫描，查找第1个排序码小于temp.key的记录*/
      if(i<j)
         pvector->record[i++] = pvector->record [j];
      while( (pvector->record[i].key<=temp.key) && (i<j) )
         i++;  /*从左到右扫描，查找第1个排序码大于temp.key的记录*/
      if(i<j)
         pvector->record[j--]= pvector->record[i];
   }
   pvector->record[i]=temp;  /*找到R0的最终位置*/
   quickSort(pvector,l, i-1);    /*找到递归处理左区间*/
   quickSort(pvector,i+1, r);   /*找到递归处理右区间*/
}
```

快速排序算法性能分析

- 当待排序记录已经排序时，算法的执行时间最长
  - 第一趟经过n-1次比较，将第一个记录定位在原来的位置上，并得到一个包括n-1个记录的子文件
  - 第二趟经过n-2次比较，将第二个记录定位在原来的位置上，并得到一个包括n-2个记录的子文件；...
  - 这样总比较次数为

$$C_{\max} = \sum_{i=1}^{n-1}(n-i) = \frac{n}{2}(n-1) \approx \frac{n^2}{2}$$

快速排序的最坏时间复杂度应为$O(n^2)$

快速排序算法性能分析

- 最好情况下，每次划分使两个子区的长度大致相等
  - 设C(n)表示对长度为n的文件快速排序所需的比较次数，则C(n)等于
    - 对长度为n 的无序区进行划分所需的比较次数n-1

    加上
    - 递归的对所得左、右两个无序子区（长度<=n/2）快速排序的比较次数
  - 快速排序的记录移动次数不大于比较次数
- 快速排序算法最好的时间复杂度为$O(n\log_2 n)$

快速排序算法性能分析

- 快速排序的平均时间复杂度是$T(n)=O(n\log_2 n)$
- 算法需要一个栈空间实现递
  - 递归栈的大小取决于递归调用的深度，最多不超过n，
  - 若每次都选较大的部分进栈，处理较短的部分，则递归深度最多不超过$\log_2 n$，所以快速排序的辅助空间为$S(n)=O(\log_2 n)$
- 快速排序算法是不稳定的

内容提要

- 排序的基本概念
- 插入排序
- 选择排序
- 交换排序
- 分配排序
- 归并排序