冒泡排序

1、 比较相邻的元素，如果第一个元素比第二个元素大，就进行交换。

2、 经过一轮遍历之后，每两个相邻的元素，重复 1 的步骤，最大放在最后面，如果有 n 个元素，那就要对比 n 次。

3、 排除已经排序出来的最大值放在后面，再进行 2 的步骤。

4、 代码实现如下：

```
def bubble_sort(alist):
    # 获取list长度,list从0开始算
    count = len(alist) - 1
    # n个元素遍历n次,（从下标0开始算）
    for i in range(count, 0, -1):
        #第i个元素和第i+1个元素进行对比
        for sub_i in range(i):
            # 大的元素冒上来
            if alist[sub_i] > alist[sub_i + 1]:
                # 如果位置i的原数比位置i+1的元素要大，则进行互换
                alist[sub_i], alist[sub_i + 1] = alist[sub_i + 1], alist[sub_i]
    return alist


list_a = [0, 12, 3, 14, 2, 1, 16]
bubble_sort(list_a)
```

当 list 有 7 元素的时候，索引最大为 6，len(list)-1 获取最后一个元素的索引。

a) 7 个元素要对比 6 次就可以找到最大值，

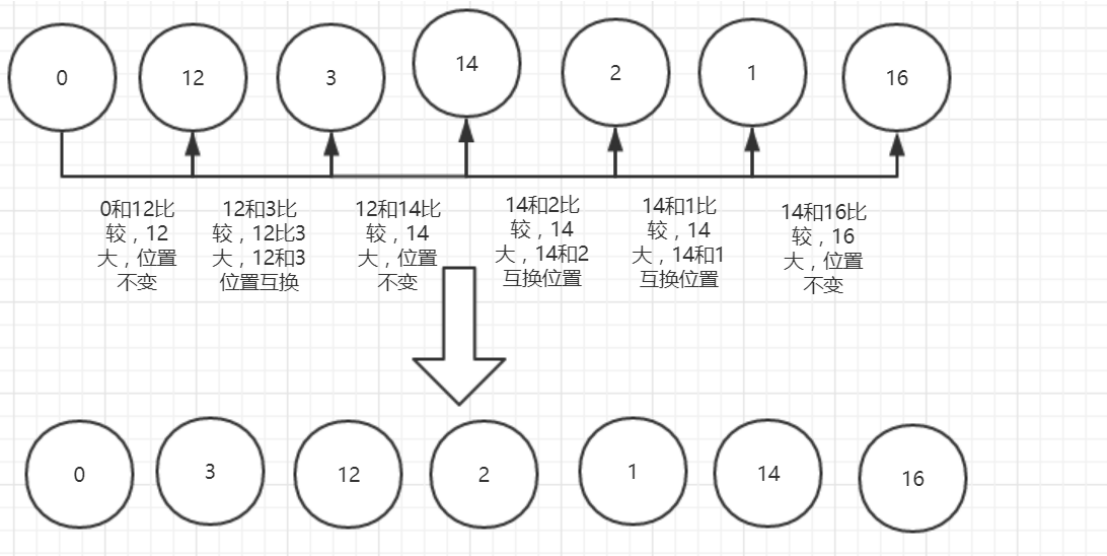
b) 还剩 6 个元素，对比 5 次找出最大值，

c) 还剩 5 个元素，对比 4 次找出最大值，

d) 还剩 4 个元素，对比 3 次找出最大值，

e) 还剩 3 个元素，对比 2 次找出最大值，

f) 还剩 2 个元素，对比 1 次找出最大值。



快速排序

1、 从列表中挑出一个元素，作为基准值 key，一般选列表最左侧的元素作为基准值。
2、 所有小于 key 的元素放在左边，所有大于 key 的元素放右边。
3、 再对左右区间进行重复 1、2 步骤，直到左右区间只有一个值。

列表 a：[10,34,12,9,23,51]

index	0	1	2	3	4	5
数组	10	34	12	9	23	51

初始化：

low = left = 0

hight = right =5

key = a[0] = 10

从 hight(5) –> low(0)寻找比 key 小的值，找到了 a[3]，然后将 a[3]插到 a[0]中，a[low]=a[hight]=9，相当于把比 key 小的值发放到 key 的左边，此时的 hight 为 3。

从 low(0) –>hight(3)寻找比 key 大的值，找到了 a[1]，将 a[1]的值插到 a[3]中，a[hight]=a[low]=34，相当于把比 key 小的值发放到 key 的右边，此时的 low 为 1。

index	0	1	2	3	4	5
数组	9	34	12	34	23	51

此时 low=1， hight=3， key=10

从 hight(3)—>low(1)寻找比 key 小的值，直到 hight=low=1 都没有找到，此时把 key 存放到 a[1]中。

index	0	1	2	3	4	5
数组	9	10	12	34	23	51

此时递归对 key 左右两侧的区间进行排练

左侧区间仅有一个元素，直接返回 9

右侧区间 a[low+1:right]，再次进行快速排列

此时， low = 2， hight=right=5， key=a[low]=12

从 hight(5) –> low(2)寻找比 key 小的值， 直到 hight=low=2 没有找到， 此时 key 的位置不变。

在进行一次递归

左侧区间不需要进行排序，直接返回

右区间进行排序

此时：low=3， hight=right=5， key=a[3]=24

从 hight(5) –>low(3)寻找比 key 小的值，找到了 a[3]，然后将 a[3]插到 a[low]中，a[low]=a[3]=23，相当于把比 key 小的值发放到 key 的左边，此时的 hight 为 4。

从 low(3)—>hight(4)中寻找比 key 大的值，知道 low=hight 都没有找到，则把 key 的值存放给 a[hight]， a[hight]=key

此时数组就变成：

index	0	1	2	3	4	5
数组	9	10	12	23	34	51

代码实现如下：

```python
def quick_sort(alist, left, right):
    # left:list最左边的位置 right:list最右边的位置
    # 左侧的位置和右测的位置一样的时候，递归停止，直接返回列表
    if left >= right:
        return alist
    # 定义游标，标记位置
    low,hight = left,right
    # 取基准值,列表最左侧的值
    key = alist[low]
    while low < hight:
        # 从右到左侧的元素依次和基准值进比较，寻找比key值小的元素
        while low < hight and alist[hight] >= key:
            hight -= 1
        # 比key小的元素放在最左边
        alist[low] = alist[hight]
        # 从左侧到右侧，依次和基准值进行比较，寻找比key值大的元素
        while low < hight and alist[low] <= key:
            low += 1
        # 比key大的元素放在最左边
        alist[hight] = alist[low]
    # 当low和height相等时补上key
    alist[hight] = key
    #处理左侧
    quick_sort(alist, left, low-1)
    # 处理右侧的位置数据
    quick_sort(alist, low+1, right)
    return alist
list_a = [10, 34, 12, 9, 23, 51]
print(quick_sort(list_a, 0, 5))
```

堆排序

1、堆排序指利用堆的数据结构设计的一种排序算法

2、堆近似一个完全二叉树结构

3、子节点的键值小于（或者大于）它的父节点，如果每个子节点值都小于它父节点，则为大顶堆，反之则为小顶堆

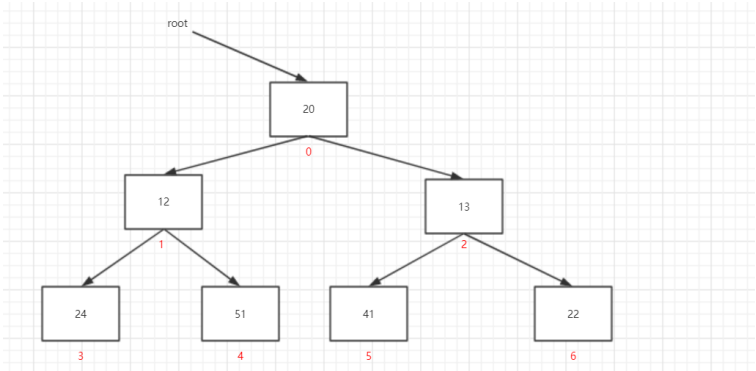
如列表 a = [20，12，13，24，51，41，22]

0	1	2	3	4	5	6
20	12	13	24	51	41	22

大顶堆：a[i] >= a[2i+1] and a[i] >= a[2i+2]

小顶堆：a[i] <= a[2i+1] and a[i] <= a[2i+2]

变成二叉树结构

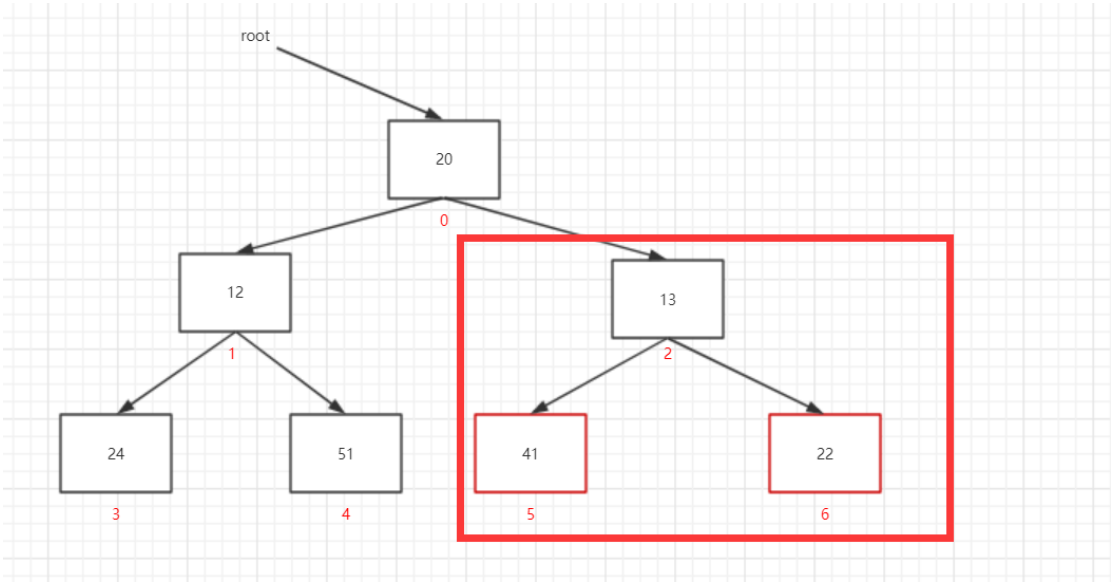


排序原理：将待排序序列构造成一个大顶堆，此时，整个序列的最大值就是堆顶的节点。将

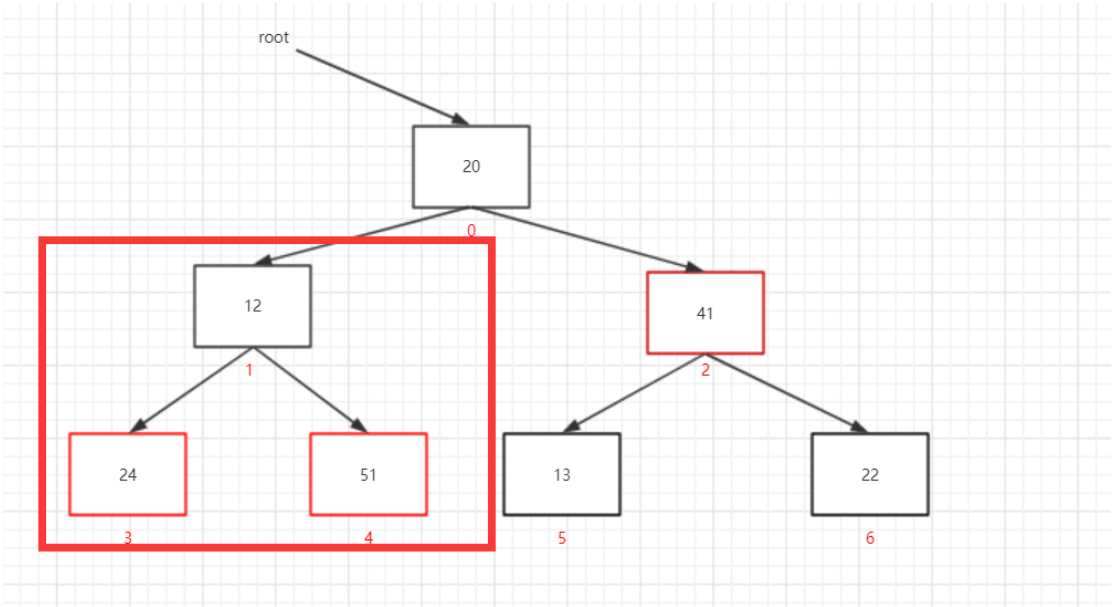
其首与尾元素进行交换，此时得到一个末尾为最大值。将剩余 n-1 个元素重新构造一个堆，得到 n 个元素的次小值，如此反复，便能得到一个有序序列。降序使用小顶堆。

二叉树简介：二叉树的每个结点至多只有二棵子树(不存在度大于 2 的结点)，二叉树的子树有左右之分，次序不能颠倒。二叉树的第 i 层至多有 2^{i-1}个结点；深度为 k 的二叉树至多有 2^k-1 个结点。

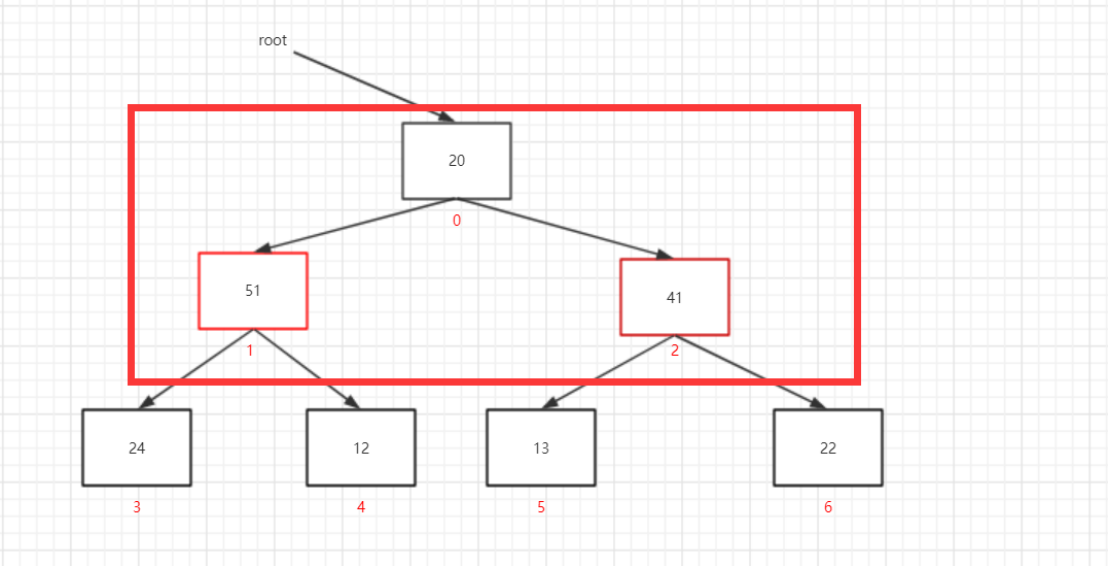
要明白的概念有：叶子：也就是没有分支的节点。如上图的 3、4、5、6 就是叶子，非叶子节点，就是该节点下面还有分支的节点，如上图的 0，1，2，其中 0 是根节点。上图有一共有三层结构，也就是说树的深度为三。
创建大顶堆的过程：
1、先从 root=2 的子节点开始比较，比较出一个大的子节点，这个子节点再和父节点进行比较。



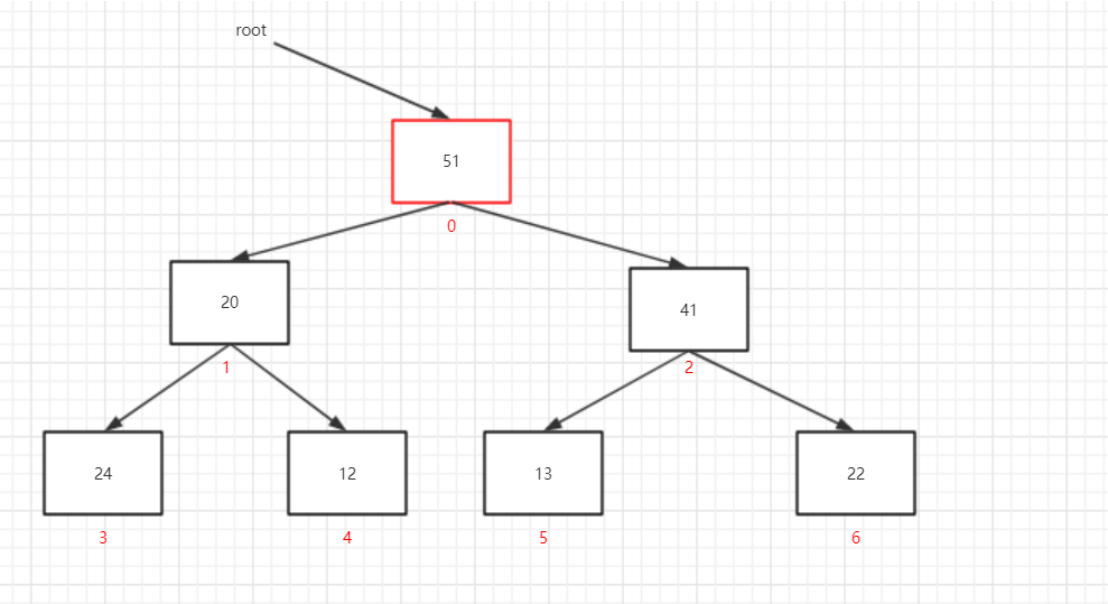
2、再到 root=1 的子节点进行比较



4、 最后 root=0 的子节点进行比较

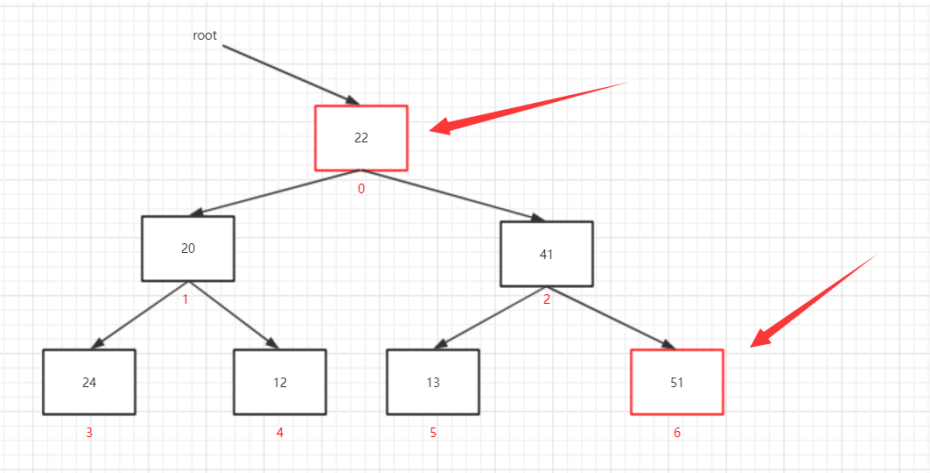


5、 最后得出大顶堆的二叉树

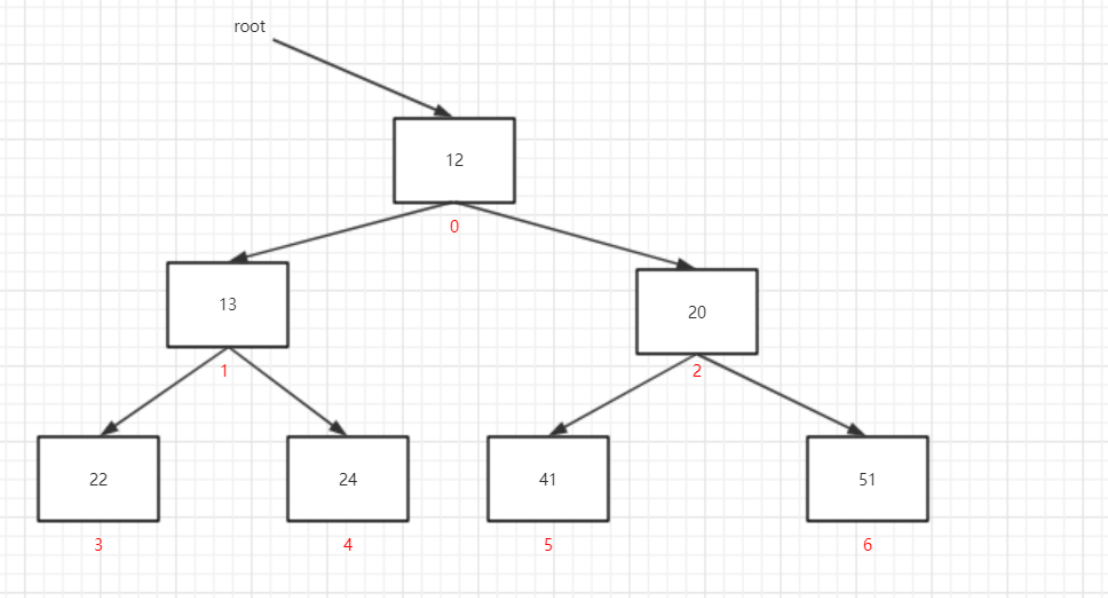


排序的过程：

1、将根节点和尾节点进行互换



2、然后剩下的元素（互换后的最后一个元素不算入其中）依然如何堆的定义，然后堆该堆再次创建大顶堆的过程，接着进行堆排序，依次类推，最后得出一个有序的序列。



此时数组为：

0	1	2	3	4	5	6
12	13	20	22	24	41	51

Python 的实现如下：

```
def heap_sort(alist):
    def sift_down(start, end):
        # 确定起始非叶子节点
        root = start
        while True:
            # 确定子节点的位置
            child = 2 * root+1
            # 终止条件，子节点的索引值超过数组的最大长度
            if child > end:
                break
            # 两个子节点进行比较大小，确定最大的子节点的索引值
            if child + 1 <= end and alist[child] < alist[child+1]:
                child += 1
            # 最大子节点和父节点进行比较，最大值变为父节点
            if alist[root] < alist[child]:
                alist[root],alist[child] = alist[child],alist[root]
                root = child
            else:
                break
    print('------------创建堆------------')
    # range((len(alist)-2)//2,-1,-1)获取的是非叶子节点
    for start in range((len(alist)-2)//2,-1,-1):
        sift_down(start,len(alist)-1)
    # print(alist)
    print('------------堆排序------------')
    for end in range(len(alist)-1,0,-1):
        alist[0],alist[end] = alist[end],alist[0]
        sift_down(0,end-1)
    return alist
```

二分查找法：

1、 二分查找又称折半查找
2、 必须采用顺序存储结构
3、 必须按照关键字大小有序排序（这个是首要条件，必须是有序的）

原理：是取一个中间值，如果要查找的值比中间值要大，则又在中间值到末端之间（中间值的右侧）取一个中间值和要查找的值进行比较，如果要查找的值比中间值要小，则又在开始端到中间值之间(中间值的左侧)再取一个中间值和要查找的值进行比较，依次类推，直到中间值为查找的值。

```python
def binary_search(arr, start, end, val):

    if start > end:
        return '出错了吧'
    mid = int((start+end)/2)
    if arr[mid] > val:
        return binary_search(arr, start, mid-1, val)
    if arr[mid] < val:
        return binary_search(arr, mid+1, end, val)
    return mid


list_a = [20, 12, 13, 24, 51, 41, 22]
#排序
list_a = sorted(list_a)
#查找
print(binary_search(list_a, 0, len(list_a)-1, 51))
```

素数原理

1、 素数即质数
2、 0、1 不是素数
3、 除了 1 和它本身外，不能被其他自然数整除的数
4、 Python 代码实现如下：（0-100）之间的所有素数

```python
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, n-1):
        if n % i == 0:
            return False
    return True

for i in range(0, 100):
    if is_prime(i):
        print(i)
```