堆排序

有一个树，请打印下面的样子

```
origin = [30, 20, 80, 40, 50, 10, 60, 70, 90] # 数据存在列表中，打印如下的样子


                    30
          20                    80
    40          50          10          60
 70    90
```

打印要求，严格对齐，就像一棵二叉树一样。

思路

第一行取1个打印，第二行取2个，第三行取3个，以此类推。如何对齐且不重叠？

代码实现

```
import math

# 居中对齐方案
def print_tree(array, unit_width=2):
    length = len(array)  # 9
    depth = math.ceil(math.log2(length + 1))  # 4

    index = 0

    width = 2 ** depth - 1  # 行宽，最深的行 15
    for i in range(depth):  # 0 1 2 3
        for j in range(2 ** i):  # 0:0 1:0,1  2:0,1,2,3  3:0~7
            # 居中打印，后面追加一个空格
            print('{:^{}}'.format(array[index], width * unit_width), end=' ' * unit
_width)
            index += 1
            if index >= length:
                break
        width = width // 2 # 居中打印宽度减半
        print() # 控制换行
```

```
# 测试
print_tree([x + 1 for x in range(29)])
```

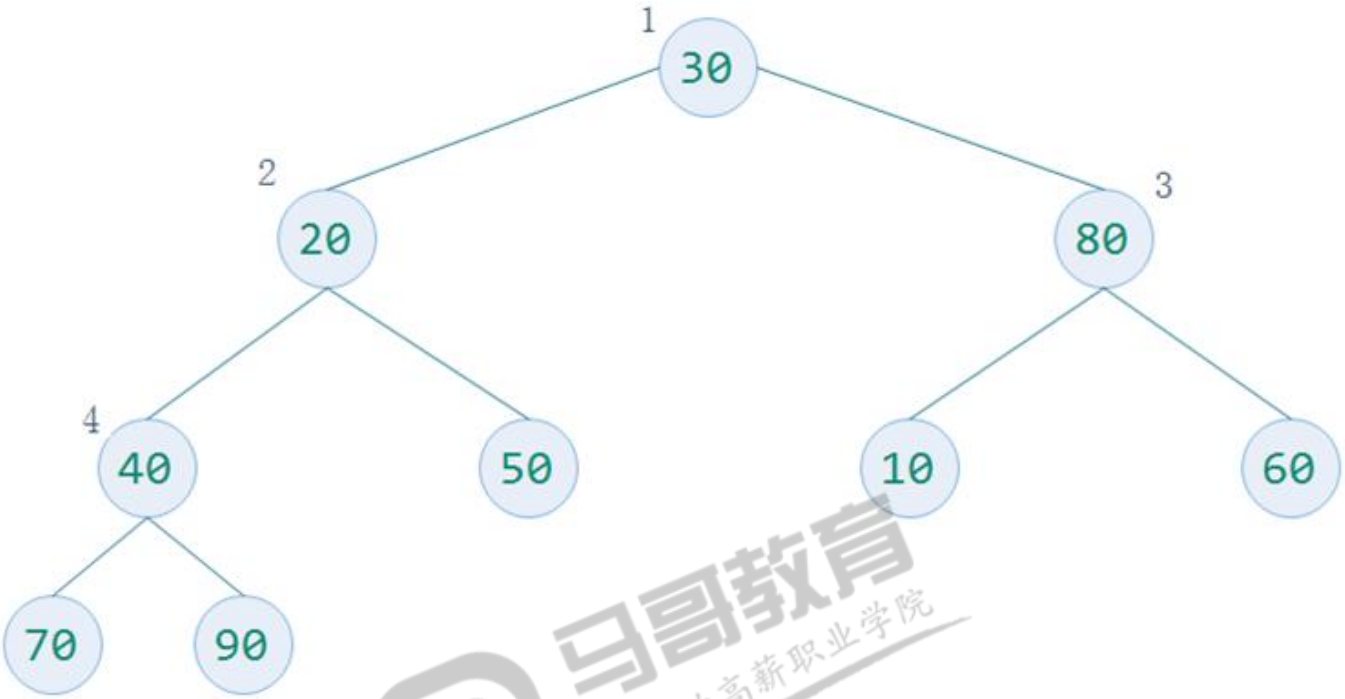
```
import math

# 投影栅格实现
def print_tree(array):
    '''
        前空格     元素间
    1    7         0
    2    3         7
    3    1         3
    4    0         1
    '''
    index = 1
    depth = math.ceil(math.log2(len(array))) # 因为使用时前面补0了，不然应该是math.ceil(math.log2(len(array)+1))
    sep = '  '
    for i in range(depth):
        offset = 2 ** i
        print(sep * (2 ** (depth - i - 1) - 1), end='')
        line = array[index:index + offset]
        for j, x in enumerate(line):
            print("{:>{}}".format(x, len(sep)), end='')
            interval = 0 if i == 0 else 2 ** (depth - i) - 1
            if j < len(line) - 1:
                print(sep * interval, end='')

        index += offset
        print()

print_tree([0, 30, 20, 80, 40, 50, 10, 60, 70, 90, 22])
print_tree([0, 30, 20, 80, 40, 50, 10, 60, 70, 90, 22, 33, 44, 55, 66, 77])
print_tree([0, 30, 20, 80, 40, 50, 10, 60, 70, 90, 22, 33, 44, 55, 66, 77, 88, 99, 11])
```

堆调整

核心算法

对于堆排序的核心算法就是堆结点的调整

1. 度数为2的结点A，如果它的左右孩子结点的最大值比它大的，将这个最大值和该结点交换
2. 度数为1的结点A，如果它的左孩子的值大于它，则交换
3. 如果结点A被交换到新的位置，还需要和其孩子结点重复上面的过程



马哥教育
IT人的高薪职业学院

```
# Heap Sort

# 为了和编码对应，增加一个无用的0在首位
# origin = [0, 50, 10, 90, 30, 70, 40, 80, 60, 20]
origin = [0, 30, 20, 80, 40, 50, 10, 60, 70, 90]

total = len(origin) - 1  # 初始待排序元素个数，即n
print(origin)
print_tree(origin)


def heap_adjust(n, i, array: list):
    '''
    调整当前结点(核心算法)

    调整的结点的起点在n//2，保证所有调整的结点都有孩子结点
    :param n: 待比较数个数
    :param i: 当前结点的下标
```

```
    :param array: 待排序数据
    :return: None
    '''
    while 2 * i <= n:
        # 孩子结点判断 2i为左孩子，2i+1为右孩子
        lchile_index = 2 * i
        # 先假定左孩子大，如果存在右孩子且大则最大孩子索引就是右孩子
        max_child_index = lchile_index  # n=2i
        if n > lchile_index and array[lchile_index + 1] > array[lchile_index]:  # n
>2i说明还有右孩子
            max_child_index = lchile_index + 1  # n=2i+1

        # 和子树的根结点比较
        if array[max_child_index] > array[i]:
            array[i], array[max_child_index] = array[max_child_index], array[i]
            i = max_child_index  # 被交换后，需要判断是否还需要调整
        else:
            break
        # print_tree(array)

heap_adjust(total, total // 2, origin)
print(origin)
print_tree(origin)
```

到目前为止也只是解决了**单个结点的调整**，下面要使用循环来依次解决比起始结点编号小的结点。

构建大顶堆

起点的选择

从最下层最右边叶子结点的父结点开始
由于构造了一个前置的0，所以编号和列表的索引正好重合
但是，元素个数等于长度减1

下一个结点

按照二叉树性质5编号的结点，从起点开始找编号逐个递减的结点，直到编号1

日哥教育 IT人的高薪职业学院

```
# 构建大顶堆、大根堆
def max_heap(total,array:list):
    for i in range(total//2,0,-1):
        heap_adjust(total,i,array)
    return array

print_tree(max_heap(total,origin))
```

思考：

```
              90
      70              80
  40      50      10      60
20  30
```

大顶堆有没有可能是这样

```
              90
      60              80
  40      50      10      70
20  30
```

有没有可能这样

```
              90
      50              80
  40      10      60      70
20  30
```

有没有可能这样？

```
              90
      50              70
  40      10      60      80
20  30
```

结论
最大的一定在第一层，第二层一定有一个次大的。

排序

思路

1. 每次都要让堆顶的元素和最后一个结点交换，然后排除最后一个元素，形成一个新的被破坏的堆。
2. 让它重新调整，调整后，堆顶一定是最大的元素。
3. 再次重复第1、2步直至剩余一个元素

```
def sort(total, array:list):
    while total > 1:
        array[1], array[total] = array[total], array[1] # 堆顶和最后一个结点交换
        total -= 1

        heap_adjust(total,1,array)
    return array

print_tree(sort(total,origin))
```

改进
如果最后剩余2个元素的时候，如果后一个结点比堆顶大，就不用调整了。

```
def sort(total, array:list):
    while total > 1:
        array[1], array[total] = array[total], array[1] # 堆顶和最后一个结点交换
        total -= 1
        if total == 2 and array[total] >= array[total-1]:
            break
        heap_adjust(total,1,array)
    return array

print_tree(sort(total,origin))
```

思考
如果有n个结点全部是90，能在哪些地方优化？
是否可以假设，如果在某一个趟排序中，如果最后一个叶子结点正好是堆顶，就代表树中元素都相等？

```
反例
    90
80      90
```

完整代码

如果有需要，请自行将算法函数封装成类。

```
import math


def print_tree(array):
    '''
        前空格     元素间
    1    7         0
    2    3         7
    3    1         3
    4   0         1
    '''
    index = 1
    depth = math.ceil(math.log2(len(array))) # 因为补0了，不然应该是math.ceil(math.log2(len(array)+1))
    sep = '  '
    for i in range(depth):
        offset = 2 ** i
        print(sep * (2 ** (depth - i - 1) - 1), end='')
        line = array[index:index + offset]
        for j, x in enumerate(line):
            print("{:>{}}".format(x, len(sep)), end='')
            interval = 0 if i == 0 else 2 ** (depth - i) - 1
            if j < len(line) - 1:
                print(sep * interval, end='')

        index += offset
        print()

# Heap Sort

# 为了和编码对应，增加一个无用的0在首位
# origin = [0, 50, 10, 90, 30, 70, 40, 80, 60, 20]
origin = [0, 30, 20, 80, 40, 50, 10, 60, 70, 90]

total = len(origin) - 1  # 初始待排序元素个数，即n
print(origin)
```

```
print_tree(origin)
print("="*50)

def heap_adjust(n, i, array: list):
    '''
    调整当前结点(核心算法)

    调整的结点的起点在n//2，保证所有调整的结点都有孩子结点
    :param n: 待比较数个数
    :param i: 当前结点的下标
    :param array: 待排序数据
    :return: None
    '''
    while 2 * i <= n:
        # 孩子结点判断 2i为左孩子，2i+1为右孩子
        lchile_index = 2 * i
        max_child_index = lchile_index  # n=2i
        if n > lchile_index and array[lchile_index + 1] > array[lchile_index]:  # n
>2i说明还有右孩子
            max_child_index = lchile_index + 1  # n=2i+1

        # 和子树的根结点比较
        if array[max_child_index] > array[i]:
            array[i], array[max_child_index] = array[max_child_index], array[i]
            i = max_child_index  # 被交换后，需要判断是否还需要调整
        else:
            break
    # print_tree(array)

# 构建大顶堆、大根堆
def max_heap(total,array:list):
    for i in range(total//2,0,-1):
        heap_adjust(total,i,array)
    return array

print_tree(max_heap(total,origin))
print("="*50)

# 排序
def sort(total, array:list):
```

```
    while total > 1:
        array[1], array[total] = array[total], array[1] # 堆顶和最后一个结点交换
        total -= 1
        if total == 2 and array[total] >= array[total-1]:
            break
        heap_adjust(total,1,array)
    return array

print_tree(sort(total,origin))
print(origin)
```