# 堆的建立与堆排序

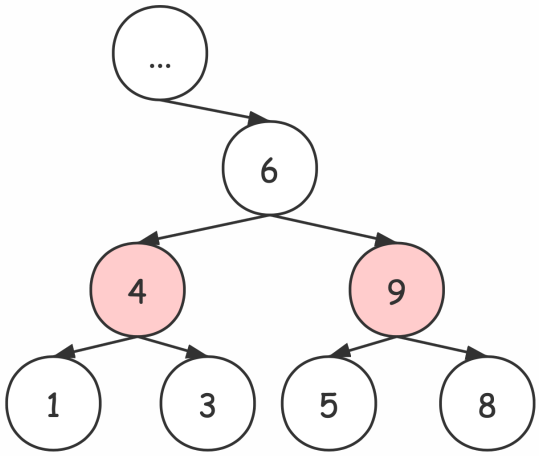
Date：2021.9.28

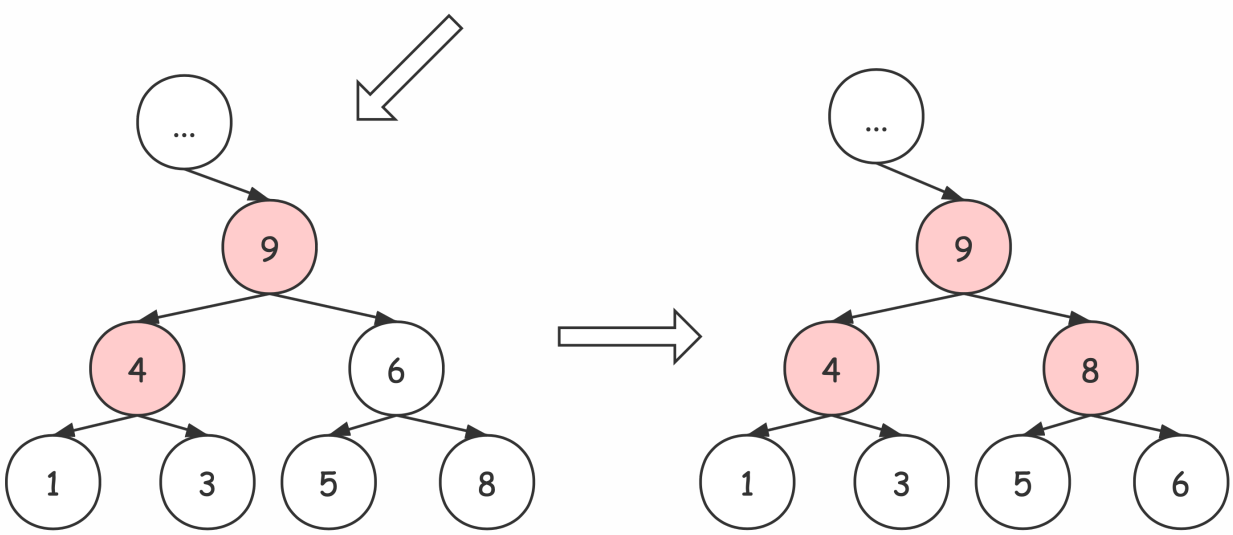
---

- 示例引出

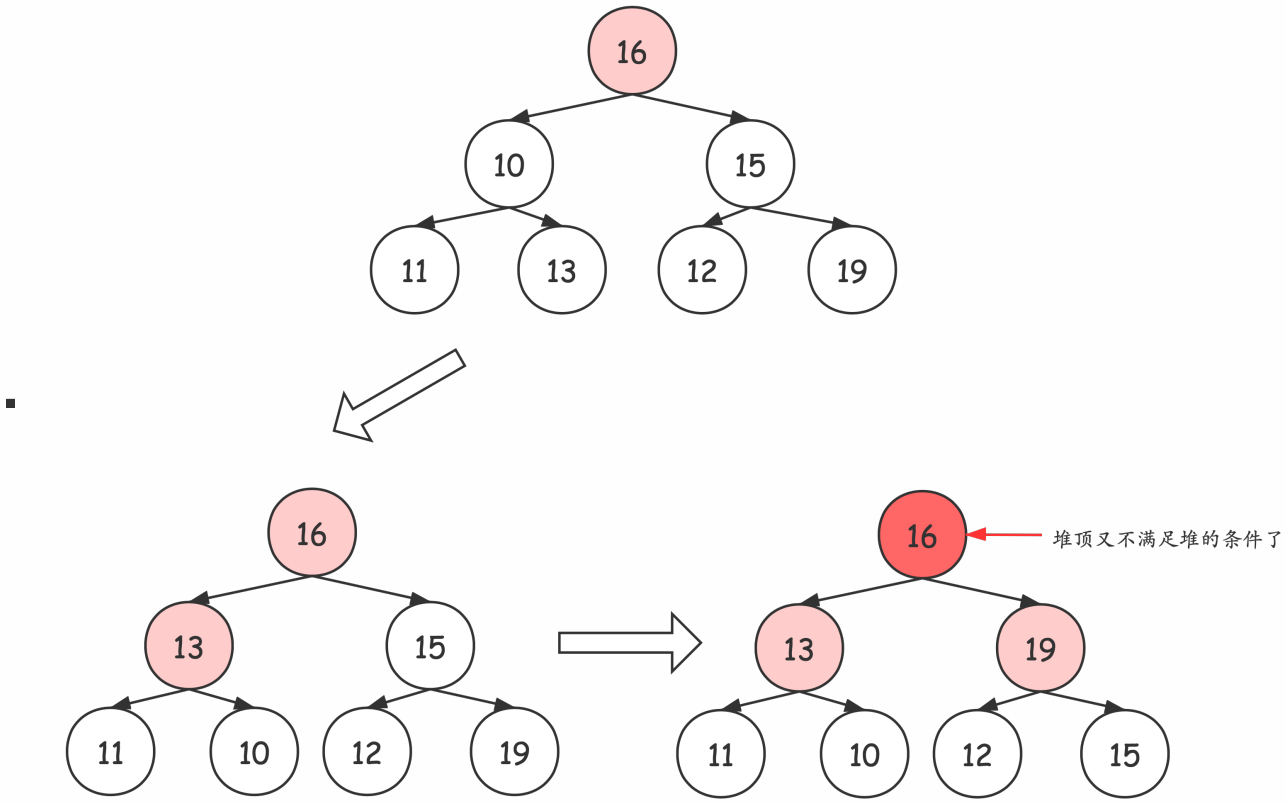
**通过数字序列 3、1、2、8、7、5、9、4、6、0 创建大根堆，并进行堆排序。**

- 创建大根堆：
  - 1. 按照正常顺序（层序顺序）读取数字序列；
  - 2. 为了建立大根堆，现在需要对堆结点从下往上，从右往左地（或者说倒序地）进行向下调整。
    - 何为向下调整？
    - 对于大根堆来说，一个含有后代的结点的权重，如果小于其后代权值中的最大者，即需要将两者相互交换；
    - 然而随着交换后结点权值的变化，原先权值最大的子结点中的权值也不一定满足原先的关系了，即结点权值一定大于其子结点中的最大权值，所以这里需要向下继续调整，直到后代全部满足关系。
    - 这里给出一个向下调整的例子，红色结点代表满足堆的条件的结点：

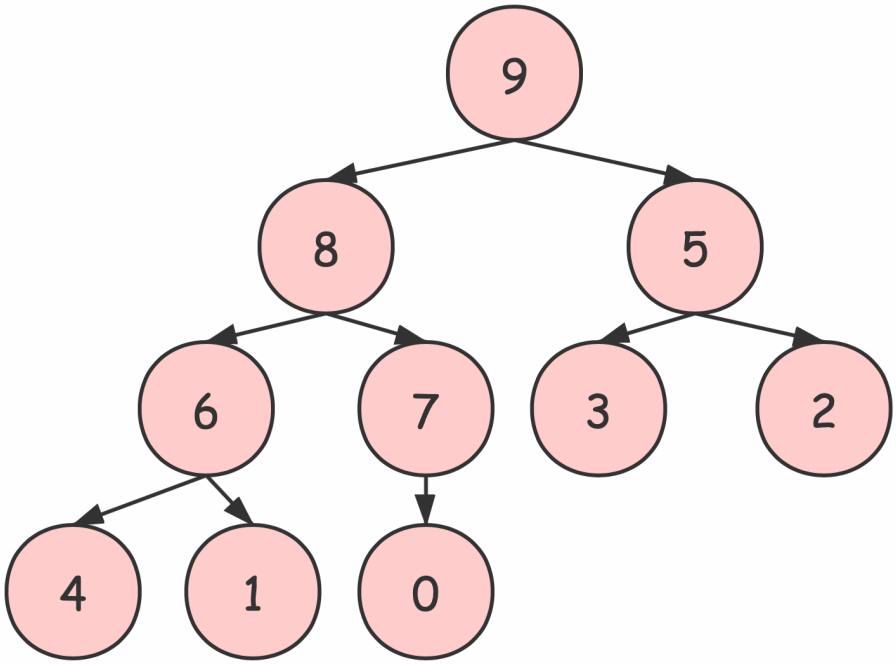



- 


- 这里还有一个问题，为什么要从下往上，倒序地进行调整呢?
  - 可以反过来思考，若我们采用从上向下顺序地进行调整，就要从根结点开始进行访问，那么，如果某个根结点的子结点都满足权值小于此根结点的权值，我们是否可以“放心地”向下继续调整呢？答案是不能，因为我们无法保证在经过之后的调整，该根结点的子结点的权值不变，换句话说，这两个子结点及其后代的权值不一定满足堆的条件，这两个子结点的权值有可能会被替换掉。而为了解决这一问题我们可能需要在每次调整之后，再向上调整一次，但是这样会造成更大的花销，得不偿失。
  - 这里我们给出一个图示来说明这一问题:




那么将上述示例的堆结构创建完成后的图示如下：




这里给出实现代码：

```
#include <iostream>
using namespace std;
int n;
int heap[110];
```

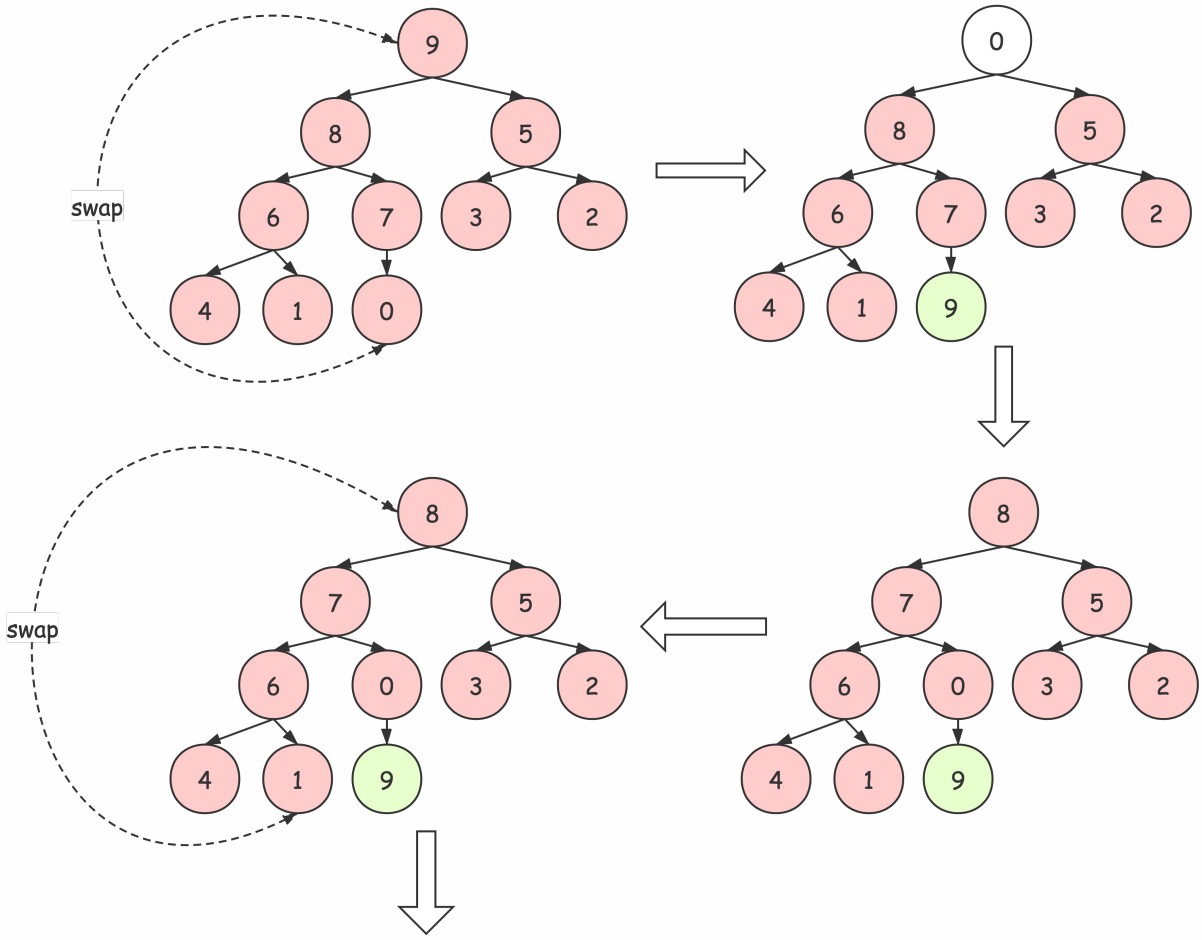
```
void swap(int &i, int &j)
{
    int temp = i;
    i = j;
    j = temp;
}

/**
 * @description: 向下调整堆结点
 * @param {int} low 堆中的非叶子结点的索引值
 * @param {int} high 当前堆中的结点数量
 */
void downAdjust(int low, int high)
{
    // 堆是一颗完全二叉树
    // 左子结点序号 = 2 * 结点索引值，右子结点序号 = 2 * 结点索引值 + 1，
    // j为待调整堆结点的左子结点索引值
    int i = low, j = low * 2;
    while(j <= high)
    {
        // 右子结点存在，选取其中最大者
        if(j + 1 <= high && heap[j + 1] > heap [j])
        {
            j = j + 1;
        }
        // 若不满足堆结点条件，则向下调整
        if(heap[i] < heap[j])
        {
            swap(heap[i], heap[j]);
            i = j;
            j = j * 2;
        }
        // 否则结束调整
        else
        {
            break;
        }
    }
    return ;
}

/**
 * @description: 按照倒序调整堆中的非叶子结点，以创建堆
 */
void createHeap()
{
    // 从最后一个非叶子结点开始
    for(int i = n / 2; i > 0; i--)
    {
        downAdjust(i, n);
    }
}
```

```
int main(void)
{
    scanf("%d",&n);
    for(int i = 1; i <= n; i++)
    {
        scanf("%d", &heap[i]);
    }
    createHeap();
    for(int i = 1; i <= n; i++)
    {
        printf("%d ",heap[i]);
    }
    return 0;
}
```

- 现在的堆已经满足了大根堆的要求，根结点权重一定大于子结点的权重，开始堆排序。
- 堆排序的流程：倒序选取元素，与堆顶元素交换，然后从堆顶向下调整，直到所有元素选取完毕。
  - 排序过程并不复杂，这里给出一次迭代的流程图示：
  -




...

- 这里给出实现代码:

```
#include <iostream>
using namespace std;
int n;
int heap[110];

void swap(int &i, int &j)
{
    int temp = i;
    i = j;
    j = temp;
}

/**
 * @description: 向下调整堆结点
 * @param {int} low 堆中的非叶子结点的索引值
 * @param {int} high 当前堆中的结点数量
 */
void downAdjust(int low, int high)
{
```

```
    // 堆是一颗完全二叉树
    // 左子结点序号 = 2 * 结点索引值，右子结点序号 = 2 * 结点索引值 + 1，
    // j为待调整堆结点的左子结点索引值
    int i = low, j = low * 2;
    while(j <= high)
    {
        // 右子结点存在，选取其中最大者
        if(j + 1 <= high && heap[j + 1] > heap [j])
        {
            j = j + 1;
        }
        // 若不满足堆结点条件，则向下调整
        if(heap[i] < heap[j])
        {
            swap(heap[i], heap[j]);
            i = j;
            j = j * 2;
        }
        // 否则结束调整
        else
        {
            break;
        }
    }
    return ;
}

/**
 * @description: 按照倒序调整堆中的非叶子结点，以创建堆
 */
void createHeap()
{
    // 从最后一个非叶子结点开始
    for(int i = n / 2; i > 0; i--)
    {
        downAdjust(i, n);
    }
}
/**
 * @description: 一次迭代，交换目标结点与堆顶结点后，从堆顶进行向下调整
 * @param {int} end 倒序访问的堆结点序号
 */
void iteration_heapSort(int end)
{
    swap(heap[end], heap[1]);
    // 这里需要使用 end - 1 来限定当前堆中能访问的边界
    downAdjust(1, end - 1);
}

int main(void)
{
    scanf("%d",&n);
```

```
    for(int i = 1; i <= n; i++)
    {
        scanf("%d", &heap[i]);
    }
    createHeap();
    // 这里可以输出每次迭代的结果
    for(int i = n; i > 0; i--)
    {
        iteration_heapSort(i);
    }
    // output heapSort result
    for(int i = 1; i <= n; i++)
    {
        printf("%d ", heap[i]);
    }
    return 0;
}
```