HeapSort

堆排序是一种基于二叉堆数据结构的，基于比较的技术。类似于选择排序，都是先找到最大的元素，然后将最大的元素放在最后。然后对剩余的元素重复相同的过程。

二叉堆

首先定义完全二叉树。完全二叉树是每一层除了最后一层都是完全满的，最后一层的节点也是尽可能的靠左。

二叉堆是一个完全二叉树，每个节点都是特殊排序的，使得父节点比它两个孩子节点都大（或小）。前者角大顶堆，后者叫小顶堆。这个堆可以用二叉树或者数组表示。

二叉堆的数组表示

因为二叉堆是完全二叉树，所以它可以被轻易地用数组表示，而且用数组表示更加空间高效。如果父节点被存在索引`i`，那么两个孩子节点是`2*i+1`和`2*i+2`，假设起始索引为`0`。

堆排算法的升序排列

- 对输入数据建一个大顶堆
- 此时堆顶是最大元素，将它和最后一个元素交换。将堆的大小减一。
- `while heap size > 1` 重复此步骤。

如何建堆

堆化（`heapify`)步骤只能被应用在一个孩子节点已经被堆化的节点上。所以堆化一定要自底向上进行。

常见的做法是，从堆大小的一半处为索引起始，向前堆化。堆化的方法可以采用下沉法。

```
    # 小的往下下沉
    def sink(self, array, n, k):
        left = 2 * k + 1
        right = 2 * k + 2
        if left >= n:
            return
        max_i = left
        if right < n and array[left] < array[right]:
            max_i = right
        if array[max_i] > array[k]:
            array[max_i], array[k] = array[k], array[max_i]
            self.sink(array, n, max_i)

    def build_heap(self, list_):
        n = len(list_)
```

```
        for i in range(n // 2 - 1, -1, -1):
            self.sink(list_, n, i)
        return list_
```

堆排的代码自然能写出

```
    def heapSort(self, arr):
        self.build_heap(arr)
        for i in range(len(arr) - 1, 0, -1):
            arr[i], arr[0] = arr[0], arr[i]
            self.sink(arr, i, 0)
```

注意点

- 堆排是in-place的排序方法
- 它典型的实现是不稳定的，但是可以使得稳定，链接。
- 时间复杂度：
    - 堆化O(logN)
    - 建堆的时间复杂度是O(N)，这里可以数学证明build_heap的时间复杂度是O(N)的。总体的时间复杂度O(NlogN)的。

堆排序的应用

- 排序一个几乎排好序的数组 (或者可以说距离目标位置不超过 K)
- k largest(or smallest) elements in an array

堆排序算法的用途有限，因为快速排序和归并排序在实践中更好（因为建堆实际上增大了数组的逆序度，而基于比较的排序的复杂度与逆序度有关）。然而，堆数据结构本身却被大量使用。