1 设计思路

堆排序的基本思路是将一个序列先排成一个大顶堆序列，此时第一个元素最大，将其取出排序，再对剩下的元素循环上述操作，为了节省空间，排序时可以将取出元素放在序列的末部。

第一个重要函数 percDown()，传递三个参数，序列 a，索引 i，大小 n. 作用是将索引 i 处的元素向下寻找合适的位置，去维护最大堆的性质。思路是先将元素 a[i] 移动到变量 tmp 中，再找左右节点较大者，与之比较，若前者大，则 break；若后者大，则将 a[i] 改为后者，更新 i 为子节点，向下循环继续寻找合适位置，再把 tmp 放入到合适的位置。

然后就可以写 heapsort() 了，传递序列 a。首先靠 percDown() 构建一个大顶堆，percDown() 是向下维护，如果对没有孙子的节点操作，我们可以保证该节点下的子树符合最大堆性质，因此我们从最后一个非叶子节点开始维护，向上遍历，就可以保证每遍历完一个节点，该节点的子树都符合最大堆性质。而最后一个非叶子节点的索引对于一个完全二叉树来说是容易得到的。之后我们进行排序，将第一个元素放到最后（相当于取出这个元素），我们再维护大顶堆的性质，此时的堆，除了第一个节点，其他节点的子树都满足最大堆性质，因此我们只需要把第一个元素向下沉，得到新的最大堆进行排序，值得注意的是，序列物理上虽然长度不变，但是每次实际上我们取走了一个元素，所以调用 percDown() 时，应当改变序列的长度。

2 测试流程

check(): 用于检查传入的序列 a 是否为升序，是即 true，否即 false。

run_time(): 用于记录 heapsort() 排序所用的时间，记时方式是 chrono。

sort_heap_time(): 用于记录 std::make_heap() 和 std::sort_heap() 排序所用的时间，和上面的差不多。

我把排序嵌入到了计时函数里面，而且保证了计时函数的参数传递方式是传值，这样就能保证两个函数是对同一个序列进行排序的计时，而不会发生第一个排序完成后，传入第二个的是排序好的序列。此外，我把 check() 也加入到了计时函数里面，如果排序没有完成，返回值将会是 0us。事实上，如果没有 std::make_heap(), 将会导致 std::sort_heap() 无法对大顶堆操作，从而导致结果为 0us, 这也是测试时发现的问题。

我用时间作为种子，生成四个长度为 1000000 的序列，分别符合 random，ordered，reverse，repetitive。其中 random 是 ordered 生成方式后打乱，reverse 是 ordered 生成方式后 reverse。

测试了几次，结果都差不多。

表 1: TIME TABLE(单位: 微秒)

	my heapsort time	**std::sort_heap time**
random sequence	104121	102301
ordered sequence	46864	42290
reverse sequence	53146	50751
repetitive sequence	111150	104702

3 结果分析

可以看出，两种排序的时间都差不多，对于一般的长度为 1000000 的序列，花了差不多 0.1 秒，而对于顺序或逆序序列，只花了 0.05 秒左右。顺序和逆序序列本身具有堆的性质，因此在排序过程中会更快。初始化大顶堆的时间是 O(N), 而后续排序维护的时间是 O(NlogN)(每取一次向下沉)，总的时间复杂度是 O(NlogN)。我另外写了一个

程序，获得了不同大小的随机序列堆排序所需时间，做了一张折线图（在文件夹中），可以看出，时间的增长是大于线性的，应证了 O(N) 的时间复杂度。