1 Heapsort 的实现

首先想要实现 **HeapSort** 大致需要实现一下几个功能: **makeheap, heapnode, deleteMin**

1.1 heapnode

为了实现 **heapnode**, 我才用了在一个 **node** 上，使用比较的方法，找出最小的一个元素并用其与 **node** 的 **root,swap** 保证最小堆的实现，再进行递归处理保证了交换后的部分以下仍是最小堆，至多进行 $O(\log n)$.

1.2 makeheap

makeheap 在对 **input.size** 进行检验是否小于等于 **1** 后，只需要着重讨论大于 **1** 的部分此时我们只需要从 **input.size/2-1** 处开始往下遍历，因为后一半的元素是没有 **children** 的. 递归结束便完成了 **makeheap.** 此时遍历后时间复杂度是 $O(n \log n)$.

1.3 deleteMin

deleteMin 实际上就是将 **root** 元素剥离然后重构 **heap**, 但为了保证 **in-place**, 我们需要将 **Input delete** 的内存给再度利用, 所以我选择使用 **swap** 第一个和最后一个元素, 最后再对 **new root** 进行堆化, 也就是 **heapnode**, 遍历整个 **heap** 后，也完成重建 **heap.**

2 Check 的实现

实际上也就是利用 **std::chrono::highresolutionclock** 函数进行运行时间的测量，同时为了保证公平性，在开始 **check** 前，我对 **input** 进行了复制，让它们进行相同的数列排序，顺序，逆序，随机，和重复四种测试，前两种我选择使用一次函数生成，后两种我选择用 **STL random** 库中的函数进行生成，并对元素范围进行限制，前者范围远大于 **input.size** 以保证随机，后者远小于 **size** 也大于 **1000**，保证可重复性. 最后再利用 **issort** 函数，对 **myheapsort** 进行检查，错误则输出 **wrong.**

3 结果及分析

我运行了数次之后，取平均值之后并保留三位小数得到：

	my heapsort time	std::sort_heap time
random sequence	0.100s	0.060s
ordered sequence	0.046s	0.028s
reverse sequence	0.045s	0.057s
repetitive sequence	0.082s	0.056s

不难发现，除了 **reverse sequence,my sortheap** 都比 **std** 效率低 (因为 **myheapsort** 是最小堆，最后实现的是逆序，于是我已经将 **mysortheap** 结果对调，以保证 **worst** 与 **best** 情况对应) 我猜测，**std** 存在一些优化策略, 比如随机化: 探测若干项是否满足序关系，如果的确如此，那么正如 **ordered** 所显示的那样，时间会大大缩短。相反如果不满足序关系占大多数，**std** 可能会先对此进行一个随机化操作，此时时间复杂度是 $O(1)$ 不会对整体复杂度产生太大影响。因此 **std** 的 **random,repetitive,reverse** 的时间近乎于相等，而我的程序却没有对此的优化，导致不同程序之间运算时间相差过大. 除了 **best** 情况剩下的均比 **std** 要慢。通过查询资料我了解到，**std** 在运算时可能有并行化向量化减少运算时间，而且同时优化内存访问模式来提高缓存命中率，这些可能也是 **std** 效率比 **myheapsort** 快的原因。