1 HeapSort 的写法

采取经典的利用堆排序进行原地排序的思路：在原 vector 先建成一个大根堆，然后依次取出堆顶并维护堆的形态。

具体的实现方法在 HeapSort.h 中可见，对于类 HeapSort，我写了如下成员函数：在 private 中：buildHeap 建堆操作，percloateDown 降堆操作；在 public 中：sort 表示堆排序。

调用的时候只需要写 HeapSort<Comparable>::sort(array) 即可，其中 array 的类型是 vector<Comparable>。

2 测试流程

我的测试范围包括五种序列：

1. 完全随机序列（Random Array）2. 有序序列（Ordered Array）3. 逆序序列（Reversed Array）4. 单一重复序列（Single-Repetitive Array，单个元素重复出现 70% 及以上的数列）5. 多重重复序列（Multi-Repetitive Array, 出现次数多于 1 的元素的总数多于 70% 的数列）

造数据方面，生成随机数使用 std::mt19937，计时使用了 std::chrono ，单位为 microseconds (1 microsecond = 1e-9 second)，序列长度取了 3×10^6。

生成重复序列的方法是：shuffle 一个下标序列获得一个置换，从而能够让重复的元素均匀分配在所有的位置上。

为了减少误差，对于每一种序列进行了五次重复实验。编译选项里面打开了 -O2 ，否则 std 速度会非常缓慢。

同时也使用了 valgrind 进行内存泄漏测试，没有发生任何内存泄漏。

3 测试结果

Sequence Type	Custom Heapsort TimeCost(microseconds)	std::sort_heap TimeCost(microseconds)
Ramdom Array	250873	246003
Ordered Array	103165	108229
Reversed Array	112118	118599
Single-Repetitive Array	106240	133842
Multi-Repetitive Array	143304	155408

表 1: 在 5 次重复实验下的测试结果

4 对于测试结果的分析

可以注意到，Custom HeapSort 实际上几乎和 std::sort_heap 的效率相当，而且有可能更快，这可能与标准库进行了过多的封装，为了通用的设计牺牲了一些效率有关，也可能与 Cache Miss 有关。