摘要

本次编程作业主要实现了堆排序。

1 设计思路

1. HeapSort 设计思路：
   定义了 `adjust(vector<T> &v, int i, const int &n)` 函数和 `sort_Heap()` 函数。`adjust()` 将 `vector v` 以 i 为根节点，长度为 n 的部分调整为最大堆。具体实现是通过小元素的下沉实现的。`sort_Heap()` 函数首先将 `vector` 变成最大堆：具体的，从最后一个有子节点的元素开始，向上一次调整。然后将最大的元素与最后一个元素交换位置，利用堆的性质，对第一个元素执行 `adjust()` 函数。

2. 测试文件设计思路:
   为了保证测试结果有一定的可重复性，设置随机数种子。分别生成了随机序列、按照升序排列的序列、按降序排列的序列和有部分重复的序列 `v`。然后用复制的方式生成一个一样的子列 `v1`，`v` 用 `std::sort_heap()` 排序，`v1` 用我写的 `sort_Heap()` 函数排序。利用 `void Checkorder(const vector<T> &v)` 这个函数来检查是否成功按照升序排列。利用 `Check` 类里的 `compareTime()` 比较运行时间。

2 程序运行结果

	my sort_Heap time	std::sort_heap time
random sequence	313 ms	502 ms
ordered sequence	233 ms	420 ms
reverse sequence	210 ms	429 ms
repetitive sequence	275 ms	465 ms

3 结果分析

可以看出我写的 `sort_Heap` 函数要比 `std::sort_heap` 函数速度快，我猜测原因如下：

1. 查询 C++ 库的实现代码可以看到

```
template <class _RandomAccessIterator, class _Compare>
inline _LIBCPP_HIDE_FROM_ABI _LIBCPP_CONSTEXPR_SINCE_CXX20 void
sort_heap(_RandomAccessIterator __first, _RandomAccessIterator __last, _Compare __comp)
```

   `std::sort_heap()` 函数不仅可以对 `vector` 的容器进行排序，也可以对其他容器里的数据排序，而我实现的 `sort_Heap()` 只针对 `vector`, 因此可以充分的利用 `vector` 的特点