

1 堆排序的设计思路

堆排序的总体思路比较简单, 根据课本算法, 即为先建堆, 再逐个取出最小(大)元即可。在 `test.cpp` 文件中, 分为前后两个部分。

1.1 借助 `algorithm` 库工具

如 Line 7 ~16 所示, 使用 `make_heap` 函数, 将 `vector` 转化为堆。然后循环使用 `pop_heap` 函数, 将最大元放到 `vector` 末尾并将剩余元素重新调整为堆。因此, `HeapSort1` 得到的序列是递增的。

1.2 自撰 `MyHeap` 类实现

如 Line 26 ~160 所示, 为了更加熟悉堆的 `insert` 与 `deleteMin` 操作, 自行撰写了简单的 `MyHeap` 类进行实现。主要包括 `public` 的 `BuildHeap`、`insert`、`deleteMin`、`PrintHeap` 函数, 以及 `private` 的 `PercolateDown`、`PercolateUp` 函数。由于建立的是最小堆, `deleteMin` 时将最小元素放至最后, 因此 `HeapSort2` 函数得到的序列是递减的。

2 测试程序

测试程序有若干重要函数组成, 主要集成各个功能, 避免重复。

2.1 测试序列的生成函数

1. `vector<int> random_vector_production()` 用于生成随机序列。
2. `vector<int> ascending_vector_production()` 用于生成递增序列。
3. `vector<int> descending_vector_production()` 用于生成递减序列。
4. `vector<int> random_repeated_vector_production()` 用于生成随机含重复元素的序列。

生成的序列长度均为 `size = 1000100 > 1000000` 符合要求。

其中, 限于所学, 生成随机序列的方法由 `gpt` 提供, 利用了随机设备和 Mersenne Twister 引擎。

随机含重复元的序列在此基础上, 直接选取一段随机序列, 多次复制到另一段随机位置, 再进行打乱实现。

2.2 检验有序函数

`bool check(const vector<int> & v)` 用于检验序列是否有序, 直接在保持未越界情况下, 比较左右两个相邻元素的大小关系即可。通过循环遍历整个序列。若存在无序情况, 则直接退出返回 `false`, 若全部符合则为有序序列, 返回 `true`。

由于 `HeapSort1` 与 `HeapSort2` 以及 `std::sort_heap` 得到的序列升降序不同, 因此考虑了升序和降序两种情况, 并在结果时取“或”逻辑, 均可通过检验。

2.3 计时函数

`void HeapSort_Check_Timing(const vector<int> & v)` 实际为最终测试的集成函数, 包含排序、计时以及判断有序性的功能。其中计时操作, 通过 `chrono` 库中的 `high_resolution_clock` 实现, 限于所学, 由 `gpt` 提供。

3 测试结果

直接输出结果如图??所示，包含排序时间及有序性检验。图中为单次结果，实际由于序列的随机性，结果会在一定范围内有所波动。

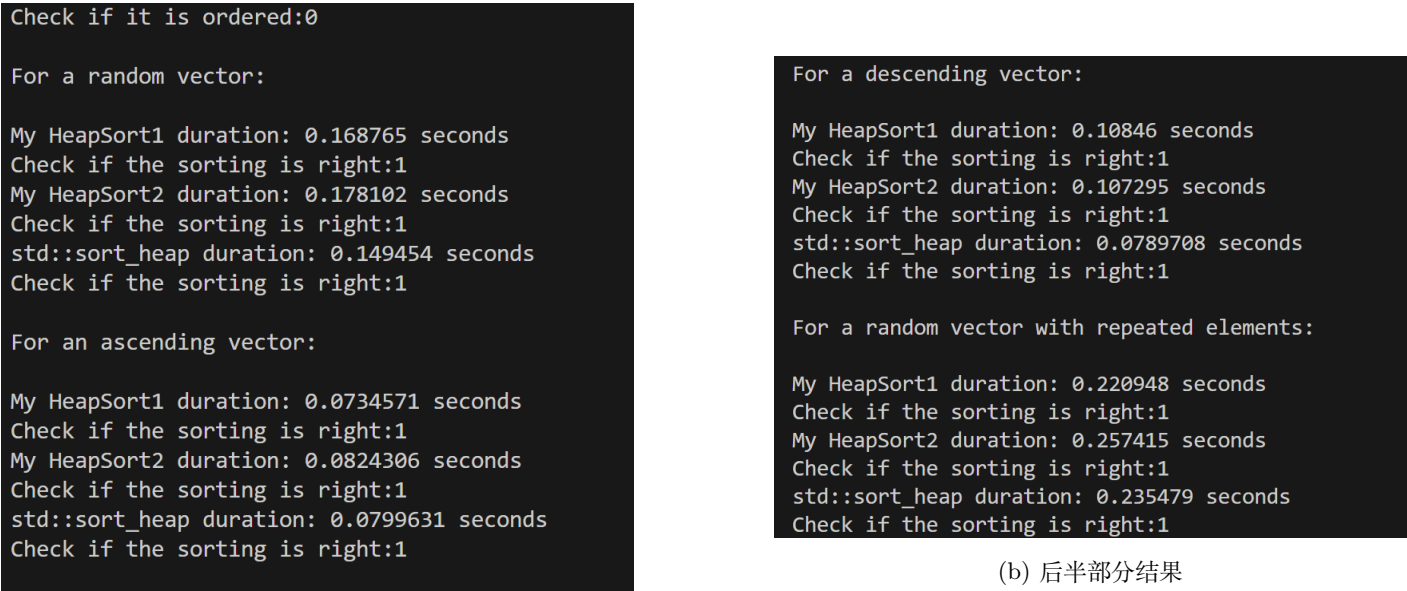


图 1: 测试结果

整理得表格如表??所示，表中单位均为 s。

| Vector Type | My HeapSort1 Duration (seconds) | My HeapSort2 Duration (seconds) | std::sort_heap Duration (seconds) |
|-------------------|---------------------------------|---------------------------------|-----------------------------------|
| Random Vector | 0.168765 | 0.178102 | 0.149454 |
| Ascending Vector | 0.0734571 | 0.0824306 | 0.0799631 |
| Descending Vector | 0.10846 | 0.107295 | 0.0789708 |
| Repetitive Vector | 0.220948 | 0.257415 | 0.235479 |

表 1: 效率对比

内存泄漏检测，如图??所示，未发现内存泄漏。利用 std 内 vector，理应自动进行内存管理，不会发生泄漏。

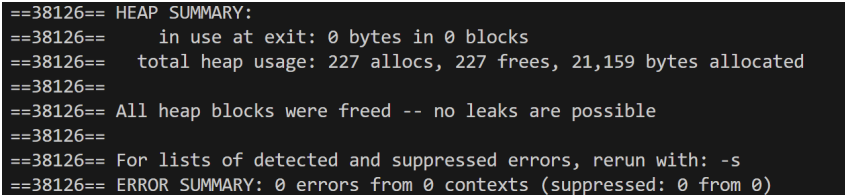


图 2: 内存泄漏检测

4 结果分析

由设计思路部分可得, `HeapSort1` 与 `HeapSort2` 均为先建堆, 建堆操作的时间复杂度为 $O(n)$, 再逐个取出最小(大)元, 取出操作的时间复杂度为 $O(\log n)$ 。因此总时间复杂度应为 $O(n \log n)$ 。复制、赋值等操作的时间复杂度均为 $O(n)$, 相比 $O(n \log n)$ 而言为小量, 故可忽略。

参考表??中的结果, `HeapSort1`、`HeapSort2` 和 `std::sort_heap` 的用时接近, 比值在 110% 到 120% 附近。说明时间复杂度应当在同一阶数, 偏差主要可能由自己撰写程序中, 一些额外的复制等操作导致, 为 $O(n)$ 量级, 为可能效率差异原因。

5 心得

在自建类 `MyHeap` 的构建过程中, 在各种函数的定义过程中会遇到一些问题, 如一些数组的防止越界, 下标等计数问题。起初出现段错误, 利用 `gdb` 调试, 从 `insert` 函数到 `deleteMin` 函数, 经过了多轮调试, 并且不断更正。同时, `test.cpp` 文件中的测试函数设计, 也不断出错调试, 整个过程还是花了很多时间。最后正确结果出来, 实在有点想要泪崩。