1 设计思路

本项目旨在实现一个基于堆排序的算法，并与标准库提供的堆排序实现（`std::sort_heap()`）进行性能对比。整个项目分为以下几部分：

1. **堆排序实现：**在`HeapSort.h`文件中实现了一个模板类，利用堆的性质对向量进行原地排序。
2. **测试程序：**在`test.cpp`中编写了一个测试程序，生成不同类型的测试序列，分别应用自定义的堆排序和标准库排序，并记录时间与正确性。

2 关键函数及实现细节

堆排序的实现主要依赖以下几个关键函数：

- `buildHeap`: 构建最大堆，从最后一个非叶子节点开始依次下滤。
- `percDown`: 下滤操作，将节点与其子节点交换以维持堆的性质。
- `sort`: 堆排序主函数，先构建最大堆，然后依次将堆顶元素与堆尾交换，最后对堆的有效部分重新下滤。

3 测试流程

写了一个check函数检测排序的正确性。同时，测试程序生成以下四种长度为1,000,000的序列：

- **随机序列：** 使用随机数生成器生成随机数。
- **有序序列：** 依次递增的整数序列。
- **逆序序列：** 依次递减的整数序列。
- **部分重复序列：** 元素为有限个值的重复序列。

每种序列分别使用自定义堆排序和`std::sort_heap`排序，记录排序时间并验证排序正确性。

4 测试结果

以下是四种序列的测试结果，包含两种排序的时间对比。

测试类型	**HeapSort时间(ms)**	**std::sort_heap时间(ms)**	效率提升
随机序列	339	506	快了33.0%
有序序列	249	375	快了33.6%
逆序序列	235	395	快了40.5%
部分重复序列	277	422	快了34.3%

5 结论

通过测试可以得出：

1. 自定义的堆排序在所有测试序列中均能正确排序，性能优于标准库的`std::sort_heap()`。
2. 在逆序序列中，自定义堆排序的效率优势最为显著。

3. 在部分重复序列中，自定义堆排序的时间也具有明显的性能提升。

整体来看，自定义堆排序是一个高效、稳定的排序实现，适用于各种场景。

6 时间复杂度分析

1. 构建最大堆的时间复杂度为O(N).

2. 下滤操作的时间复杂度为O(logN).

3. 于是堆排序的时间复杂度为O(N+NlogN)=O(NlogN)