1. Huffman Tree Using PQ **12A1**

在示例代码包中，我们基于不同的数据结构分别实现了Huffman算法。具体来说，该算法底层的数据结构——即HuffmanForest的实现方式——至少可有List、ComplHeap和LeftHeap三种选择。

a) 在掌握了ComplHeap和LeftHeap之后，试找到三个对应的项目，分别编译、运行和测试；

b) 确认这三个项目均采用了**统一的**算法框架，也就是说，对应的源代码实际上是同一个文件；

c) 三种实现方式在**各方面**的性能有何差异？

2. AVL As PQ **12A2**

a) 试基于AVL树，通过派生来实现PQ。

b) 这种实现方式的空间复杂度是多少？各接口的时间复杂度如何？

3. Vector/List As PQ **12A2**

本节简略介绍了基于无序、有序的向量、列表实现PQ的方式，并列出了主要接口对应的效率。如果只限于这四种实现方式，在实际应用中你会更加倾向于哪种方式？为什么？

4. Multiple Inheritance **12B1**

学习或温习C++的**多重继承**特性，理解这里如何利用这一特性，由PQ和Vector派生出ComplHeap。

5. insert() & percolateUp() **12B2**

课上已简要说明，在关键码均匀分布的条件下，完全二叉堆的插入操作期望地只需常数时间。试估算一下这个**常数**大致是多少。

6. percolateUp() **12B2**

上滤的过程无非是一系列的交换操作，进一步地不难看出，参与各次交换的总有一个元素是**固定的**——即被插入/上滤的那个元素。试基于这个事实调整示例代码，使该元素在全程中只需移动**一次**而非$\mathcal{O}(\log n)$次。该算法的渐近复杂度，是否因此会有所改进？

7. remove() & percolateDown() **12B3**

试证明，即便关键码是均匀分布的，完全二叉堆的删除操作期望地仍需要$\mathcal{O}(\log n)$时间。

8. percolateDown() **12B3**

与上滤类似地，下滤过程中所有交换操作中也总有一个元素是**固定的**。试参照此前对上滤算法的改进，减少元素移动的总次数。

9. heapify() **12B4**

Floyd算法可以保证在$\mathcal{O}(n)$时间内，将n个元素组织为一个完全二叉堆。但就**期望**复杂度而言，直接调用n次insert()的朴素方法也是$\mathcal{O}(n)$。从**常系数**角度来看，两种方法的优劣如何？

10. heapify() **12B4**

a) 试证明，Floyd算法中for循环的条件判断“-1 != i”可以等效地改作“i < n”；

b) 为什么我们还是倾向于前一种实现方式？

11. Cache **12B**

我们知道，ComplHeap在底层实质上是借助一个向量来存放所有元素的，那么这是否意味着，系统缓存因此可以得到**充分**利用？为什么？

12. Kruskal Using ComplHeap **12B**

我们知道，Kruskal算法需要从短到长地逐条尝试图中各边。为此固然可以先对所有的边按长度**排序**，但$\mathcal{O}(e \log e) = \mathcal{O}(e \log n)$的“投资”往往失之鲁莽。如果改用小顶的ComplHeap仅维护一个**偏序**，则这方面的时间成本可以（借助Floyd算法）降至$\mathcal{O}(n)$；当然，从后每次还需再花费$\mathcal{O}(\log n)$时间取出下一条边。

a) 试分别按两种方式实现Kruskal算法；

b) 两个算法在时间性能方面的相对优劣，主要取决于哪个（些）因素？

13. insert() & remove() **12B**

a) 我们在此前已看到，PQ::insert()算法的**期望**时间复杂度可能降至$\mathcal{O}(1)$。
试自行查阅资料，了解二项式堆、Fibonacci堆，尤其是确认它们insert()算法的时间复杂度。

b) 以上堆的各变种的remove()算法，时间复杂度各是多少？

c) 它们的remove()算法可否如insert()那样，至少在**期望**的意义下能够低于$\Omega(\log n)$？为什么？

14. Double-Ended Priority Queue **12B**

a) 所谓的**双端优先级队列**，在insert()之外，还同时兼顾delMax()和delMin()接口。该ADT最原始的一种实现方式，无非是同时维护“对称的”两个堆：一个大顶堆，另一个小顶。
这种实现方式的时间、空间复杂度是多少？主要的缺点在哪里？

b) DEPQ有多种更为高效的实现方式，比如**对称最小最大堆**（SMMH，Symmetric Min-Max Heap），仅使用一个树形结构，每个元素在其中只有一份记录。试搜索并查阅相应的材料，理解其原理及算法过程。

c) 在示例代码包中找到对应的项目，阅读代码，并编译、运行和测试。

15. Heapsort **12C**

如果**调转**方向，将待/已排序的元素放在前/后缀，在实现堆排序算法时会有什么问题？

16. Heapsort **12C**

讲义中的堆排序算法追求形式上的简洁，试改写该算法，以（在常系数意义上）减少元素**移动**的次数。

17. Replay In A Winner Tree **12D2**

试具体地实现**胜者树**结构：说明各组成部分及其关系，并针对**重赛**算法做一解释和分析。

18. Winner Tree vs. ComplHeap **12D1**

a) 为了从10,000个整数中选出最大的5个，试分别基于**胜者树**、**完全二叉堆**各设计并实现一个算法。

b) 试从空间消耗量、时间成本的角度，对两个算法做一分析对比。

19. Replay In A Loser Tree **12D2**

试具体地实现**败者树**结构：说明各组成部分及其关系，并针对**重赛**算法做一解释和分析。

20. d-Ary Heap **12E**

采用多叉堆来优化PFS算法的理论依据是，下滤操作通常都**远少于**上滤。具体地，本节分别用顶点数n、边

数e来估计这两种操作的次数。然而实际上，尽管前一估计是准确的，但后一估计只是一个上界：并非每次调用prioUpdater()都会引发上滤；即便引发上滤，节点上升的高度也未必达到$\mathcal{O}(\log n)$。

a) 鉴于以上原因，本节针对分叉数所给的建议值 d = e/n + 2 相对于**最优值**是偏大还是偏小了？

b) 你觉得可以从哪些方面着手，**更加**精准地做出估计？

21. Multiple Inheritance 12F3

学习或温习C++的**多重继承**特性，理解这里如何利用这一特性，由PQ和BinTree派生出LeftHeap。

22. LeftHeap::merge() 12F3

a) 递归版、递归版中的短路求值和特判，分别针对什么特殊情况？试分别给出具体的实例。

b) 迭代版首先通过**特判**，处理了堆a或/和b为空的情况。如果只是用merge()来实现PQ的标准接口insert()和delMax()，这些特判是否必要？为什么？

c) 迭代版的二路归并过程中，堆a、堆b中节点的**归入方式**很不相同，试对照代码分析体会。

d) 以本节针对递归版的实例，对照迭代版做一推演。两种合并实现方式的结果是否一致？

23. Degeneracy 12F4

左式堆的insert()、delMax()可以如讲义所示，统一地借助merge()来实现。

a) 试通过分析进一步确认，即便是在**退化情况**——比如插入首个元素、删除最后一个元素——下，这里的实现方式也是安全且正确的；

b) 试在示例代码包中找到对应的项目，通过**实际测试**验证上述分析结论。

24. LeftHeap --> Fibonacci Heap 12F

所谓**Fibonacci堆**可以认为是左式堆的改进，主要是改用了新的**上滤算法**，并采用**懒惰合并**的策略。

a) 试查找并阅读相关资料，自学这种数据结构。

b) 就**分摊的**意义而言，该结构的insert()、merge()、increase()等接口的时间复杂度都是$\mathcal{O}(1)$的，而delMax()却仍是$\mathcal{O}(\log n)$，你认为后者是否还有改进余地？为什么？