1. Static/dynamic operation 03A

我们知道，视其是否会改变数据集的内容及结构，对数据结构的操作可分为**静态**、**动态**两种。

a) 然而如果数据集中各元素的**逻辑**结构与**物理**结构未必一致，此时的静态、动态操作又当如何区分？

b) 你能举出这类数据结构的一例吗？

2. Call-by-rank/position 03B

a) 秩与位置之间有何联系与区别？

b) 通过重载操作符“[]”可令列表模仿向量的“循秩访问”方式，但我们为何**不建议**经常使用这种方式？

3. Sentinel 03C[1-4]

a) 引入哨兵header、trailer之后，列表主要接口的代码逻辑在哪些方面更为**精简**了？

b) 运行**时间**方面，这些接口各有多大的改进？

c) 针对本节介绍的各种列表接口，验证借助哨兵的确可以正确处理各种**边界**情况（比如空表）。

4. insert/move() 03C1

本节实现的节点插入、删除算法中，都有若干对pred、succ的调整。这些调整的**次序**是否可以调换？

5. copyNode() 03C2

a) 试在构造时调转方向，改用insertAsFirst()而非insertAsLast()来实现List::copyNode()，并依然可以保证新列表中的所有节点，能够保持原先的相对**次序**；

b) 如此调整之后，算法的**时间**、**空间**性能有何变化？

6. clear() 03C2

a) 试在析构时调转方向，改为通过反复调用remove(trailer->pred)而非remove(header->succ)来实现List::clear()；

b) 如此调整之后，算法的**时间**、**空间**性能有何变化？

7. deduplicate() 03C3

a) 该算法的循环不变性为“始终维护一个无重的**前缀**”，如果改为**后缀**，又当如何实现该算法？效率如何？

b) 该算法在前缀中一旦发现与节点p**相等**的q，便随即删除q——如果反过来删除p，代码需如何调整？

c) 自行设计若干测例，验证该算法的正确性，尤其能正确处置各种**边界**情况；

d) 试按照其它思路，设计和实现这个接口，并做实测对比。

8. traverse(): Basic 03C4

a) 对照Vector::traverse()和List::traverse()接口，理解通过**函数指针**或**函数对象**的实现方式；

b) 在实例代码包中，找到并阅读increase_Elem.h、decrease_Elem.h、double_Elem.h、half_Elem.h等源文件，了解如何通过**函数对象**来定义对单个元素的基本操作；

c) 阅读increase_vector.h、double_vector.h等源文件，理解如何将上述自定义的基本操作**顺序地施加至**向量中的每个元素，以实现对向量的遍历；

d) 阅读decrease_list.h、half_list.h等源文件，理解如何将上述自定义的基本操作**顺序地施加至**列表中的每个元素，以实现对列表遍历；

e) 模仿上述套路定义自己的基本操作，并相应地实现对向量、列表的遍历。

9. traverse(): Advanced 03C4

a) 阅读crc_Elem.h，理解如何通过**函数对象**的构造函数，以参数形式传入、传出全局性信息；

b) 阅读crc_list.h、crc_vector.h，了解如何通过**遍历接口**来计算序列的CRC校验码（这里简化为总和）；

c) 阅读checkOrder_Elem.h，理解如何通过**函数对象**的构造函数，以参数形式传入、传出全局性信息；

d) 阅读checkOrder_list.h、checkOrder_vector.h，了解如何通过**遍历接口**来判定序列是否有序；

e) 模仿上述套路定义自己的（涉及全局性信息的）基本操作，并相应地实现对向量、列表的遍历。

10. uniquify() 03D1

a) 该算法起始处，为何需要对平凡的列表（_size < 2）做**特判**？

b) 你觉得可以如何改写，使得这段代码的逻辑更为**简洁**？

c) 自行设计若干测例，验证该算法的正确性，尤其能正确处置各种**边界**情况；

d) 试按照其它思路，设计和实现这个接口，并做实测对比。

11. search(): failure case 03D2

a) 试确认，该算法在失败时可能会返回header并进而造成**误判**；

b) 你觉得可以如何改写，以**消除**这种误判？

12. search(): efficiency 03D2

a) 试确认，即便列表已经**有序**，运用目前已讲授的技术仍不能有效地降低search()接口的复杂度；

b) 这一现象背后的根本**原因**，你认为是什么？

13. selectionSort() 03E

如果注意到在该算法初始化阶段，已经令：

```
head = p->pred;
```

且p在整个算法过程中始终未被修改过，你或许会考虑将其中的：

```
insert( remove( selectMax( head->succ, n ) ), tail );
```

改为：

```
insert( remove( selectMax( p, n ) ), tail );
```

a) 在示例代码包中找到对应的工程，编译运行list的测试程序；

b) 做如上调整后重新运行，会出现什么问题？为什么？

14. selectionSort() by move() 03E

正如课上所指出的，讲义上实现的选择排序算法虽外观上逻辑简洁，但实际上运行的效率并不算高。

a) 试理解这背后的原因；

b) 试增加List::move()接口，在列表中用$\mathcal{O}(1)$时间**移动**一个节点；

c) 如果要能为List::selectionSort()所用，该接口可以/应该包含哪些参数？

d) 试实现这个接口，完成测试，并与该算法的原版本就性能做一实测对比；

15. selectionSort() by exch() 03E

在选择排序中对无序前缀中的最大元素，既可以采用**平移法**将其转入有序后缀，也可以采用**交换法**。

a) 试增加List::exch()接口，在列表中用$\mathcal{O}(1)$时间**交换**一对节点；

b) 如果要能为List::selectionSort()所用，该接口可以/应该包含哪些参数？

c) 试实现这个接口，完成测试，并与List::selectionSort()原版本就性能做一实测对比。

16. Expected #Cycle **03F**

考查对长度为n的随机序列做选择排序，试验证：

a) 每次将无序前缀中的最大元素M与末元素T交换之前，M和T应当**同属**一个循环节，且二者在其中**紧邻**；

b) 若无序前缀的长度为r，则M与T系**同一**元素（即M所属的循环节长度为1）的概率为$1/r$；

c) 循环节的**期望**总数为$\mathcal{O}(\log n)$。

17. Swap vs. Cycle **03F**

a) 在一个序列中交换一对元素，序列中的哪些循环节会有**变化**？

b) 变化可能有几种**情况**？试逐一列举并说明。

18. insertionSort() vs. selectionSort() **03G**

a) 我们知道插入排序属于**在线**算法，你所知道的其他排序算法中还有哪些也是？

b) 在玩纸牌游戏时若希望手中的牌始终顺序排列，你在抓牌过程中为何会采用**插入**排序而非其它算法？

c) 我们知道插入排序属于**输入敏感**算法，你所知道的其他排序算法中还有哪些也是？

d) 我们知道**选择**排序既非在线的，也没有最好情况，那么反过来，它相对于**插入**排序有何优势？

19. insertSort/selectionSort() using vector/list **03G**

插入排序、选择排序均可基于向量或列表实现，相对而言，二者分别**更适合**于哪种实现方式？

20. insertionSort() by move() **03G**

正如课上所指出的，讲义上实现的插入排序算法虽外观上逻辑简洁，但实际上运行的效率并不算高。

a) 试理解这背后的原因；

b) 试增加List::move()接口，在列表中用$\mathcal{O}(1)$时间**移动**一个节点；

c) 如果要能为List::insertionSort()所用，该接口可以/应该包含哪些参数？

d) 试实现这个接口，完成测试，并与该算法的原版本就性能做一实测对比；

21. Best/average case of insertionSort() **03G**

a) 我们知道，当输入序列业已有序时插入排序只需运行$\mathcal{O}(n)$时间，这种最好情况出现的概率有多大？

b) 新元素被插入有序前缀后，有可能其实原地未动。期望而言，在整个排序过程中这种情况会出现多少次？

c) 平均而言，插入排序需要运行多少时间？

22. search() in insertionSort() **03G**

我们知道，插入排序算法在插入每个新元素之前，都要在有序前缀中查找确定合适的位置。

a) 既然此处被查找的前缀始终都**有序**，为何不将**顺序查找**改作**二分查找**？

b) 如果必须改为**二分查找**，存储有序前缀的数据结构需要哪些接口？各接口的效率需要有多高？

23. Complexity of 2-way merge **03H**

a) 如果待归并的列表长度不等甚至相差悬殊，二路归并算法的运行时间还能保持线性吗？

b) 一般地，若二者的长度分别为n和m，运行时间应是多少？

24. 2-way merge by move() 03H

在课上实现的List::merge()算法中，待归并子列表中的每个元素，都需先经remove()摘除再通过insert()插至最终列表。同样地，这种方式固然形式简洁、操作安全，但效率上会有所折扣。

a) 如果能通过如下接口：

```
List::move( ListNodePosi<T> q, List<T> L, ListNodePosi<T> p );
```

将当前列表中的节点q，直接转移至另一列表L中的节点p之前，你会如何改写二路归并算法？

b) 试设计并实现这样一个操作接口；

c) 为不致破坏对L的封装，你采用了什么技术方法？

25. Vector/List::mergeSort() 03H

基于向量或列表来实现归并排序，各有什么优势与不足？

26. Vector/List::mergeSort() 03H

我们知道，无论是基于向量或列表来实现归并排序，只要如讲义那样始终以中点来切分子任务（子序列），就能保证总体的时间复杂度为$\mathcal{O}(n \log n)$。

a) 如果改为总是按 $\mathbf{2/3 : 1/3}$ 的比例来切分呢？

b) 改用 $\mathbf{15/16 : 1/16}$ 的比例呢？

c) 一般地，改用 $\lambda : \mathbf{1} - \lambda$ 的比例呢？

d) 如果总是按 $\mathbf{n - 1 : 1}$ 来切分，那么mergeSort()相当于**退化**成了哪种排序算法？

27. #inversion 03I

a) 试确认，一个序列中所含的**逆序对**总数可能多达$\Omega(n^2)$，并给出这样的实例；

b) 就数学**期望**而言，一个长度为n的序列中会含有多少个逆序对？

c) 根据上一结论，插入排序算法的**平均**时间复杂度应是多少？

28. #inversion 03I

设一个长度为2n的向量，用02E节的bubblesort()提前终止版，经过n轮扫描交换便完成了排序。

a) 该向量**最少**、**最多**可能包含多少个逆序对？

b) 若n = 15，试分别给出一个对应的整数向量实例。

29. Inversion & Exchange 03I

a) 在序列中交换一对逆序元素，逆序对的**总数**会如何变化？可能有哪几种**情况**？

b) 如果可以选择，在排序过程中我们为何倾向于优先交换相距**更远**的逆序对？

c) 试证明：一个序列中的逆序对无论多少，我们都只需$\mathcal{O}(n)$次**交换**便可使之有序；

d) 你所知的排序算法中，哪些可以保证所做的交换操作**不超过**$\mathcal{O}(n)$次？哪些不能保证？

30. Counting inversions 03I

a) 试确认，借助归并排序必能在$\mathcal{O}(n \log n)$时间内**统计**出任一序列中的逆序对总数；

b) 你还有什么其它办法也可以完成这一统计？

c） 有没有**更加高效**的办法?

31. Cursor list 03J

a） 在什么情况下，我们需要以游标模式来实现（或者更准确地，来模拟）列表?

b） 对讲义中的实例，理解这种结构的原理，推敲其运作过程，并尝试实现该结构。

32. Java Vector/List 03XA

a） 了解Java语言中interface、implements等关键词的含义与作用;

b） 在示例代码包中找到用Java描述和实现的数据结构及算法;

c） 阅读其中Vector和List的代码，与C++的实现做一对比;

d） 运行自带的测试程序，理解测试过程及结果。

33. Python List 03XB

a） 在示例代码包中找到用Python描述和实现的数据结构及算法;

b） 阅读源文件reverse.py，理解并对比其中那两个reverse()算法;

c） 了解这里所用的测试方法，运行并观察统计结果。