

**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) 试调转方向，改用insertAsFirst()来实现List::copyNode();
- b) 如此调整之后，算法的时间、空间性能有何变化？

**6. clear()****03C2**

- a) 试调转方向，改为通过反复remove(trailer->pred)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重复的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查找接口的复杂度;
- 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 $O(1)$ 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 $O(1)$ 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 $T$ 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 $O(\log n)$ 。

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

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? 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) 在玩卡牌游戏时若希望使抓到的牌顺序排列, 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 $O(1)$ 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, 当输入序列已有序时插入排序只需运行 $O(n)$ 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. #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？

**27. #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。

**28. 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 $O(n)$ 次交换便可使之有序；

d) 你所知的排序算法中，哪些具有上述特性c)？哪些不具有？

**29. Counting inversions****03I**

a) 试确认，借助归并排序必能在 $O(n \log n)$ 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？

**30. 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。

**31. 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。

**32. 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。