**《程序设计专题》实验报告2**

**链表、排序与模块化程序设计**

## 1．实验目的

通过实验：

1）掌握大程序中的函数与程序结构设计思想方法；

2）掌握链表的基本操作；

3）掌握基本的排序算法。

## 实验内容

1. 多文件实现基于链表结构的整数排序算法工具集
2. 打包本实验报告(转为pdf)、源代码，压缩为.zip格式，上传至学在浙大。上传的压缩文件命名为学号\_姓名\_HW2.zip。

## 3. 实验记录解答

**整数排序算法工具集**

**采用链表结构实现以下排序算法**：

* 简单选择排序算法
* 冒泡排序算法
* 插入排序算法

**采用多文件的程序设计方法，比如可以分为以下模块**：

* list.c/h：链表工具模块，实现基本的链表操作，比如插入、删除、交换等；
* main.c/h：主函数模块，读取输入，选择相应的排序算法，打印排序时间，打印排序后的序列等；
* sort.c/h：排序算法工具模块，实现各个排序算法。

文件模块设计内容包括介绍每个模块完成的功能、维护的数据结构（如果有），实现过程描述各个模块对数据的操作（算法），算法可以用伪代码形式来体现，或用语言来描述。

源代码中应当有注释，以方便它人阅读和理解。注意代码规范，例如缩进、变量命名、函数命名等。

**解答：**

**3.1 文件模块设计**

**main.c：**主程序，首先选择用作测试的排序算法，从标准输入或文件读入排序数据，调用选定的排序算法，统计排序时间，输出排序后的结果。

**general.h：**全局头文件，包含了常用的标准库头文件，定义了链表结构。

**list.c/h：**实现了链表的创建、销毁、打印、复制等操作。

**sort.h：**定义了sort\_t结构体用于存放一种排序算法的名称和函数指针，便于主程序循环调用。

**sort.c：**除了实现规定的选择排序、冒泡排序和插入排序外，还自行增加了类选择排序、归并排序和标准库的快速排序，以便更好比较排序算法。并声明了sort\_t数组。

**Makefile：**使用Make构建程序。

**generate.c：**用于生成大规模的随机排序数据。

**3.2 实现过程**

**main.c：**首先选择用作测试的排序算法，处理命令行参数，重定向标准输入输出，用于大规模数据的测试。然后读入数据，对每种排序算法，复制一份数据用于排序并打印，用POSIX标准的高精度计时器计时，最后释放复制的链表。

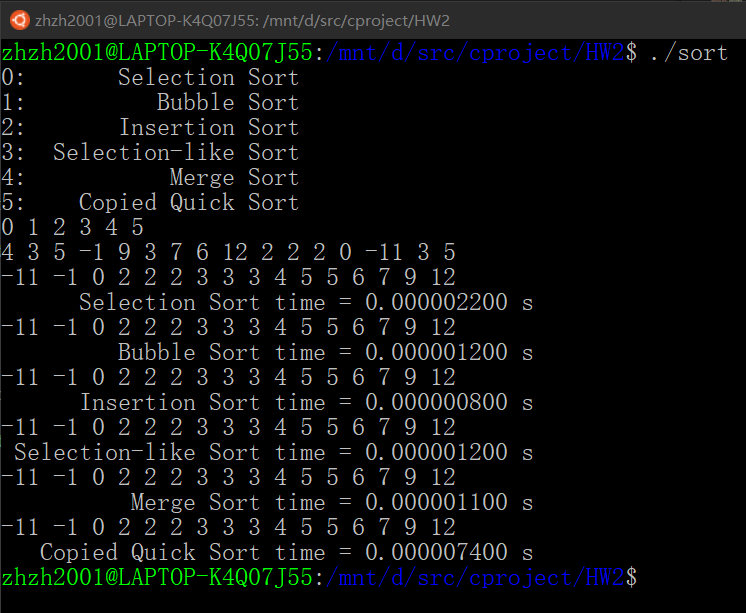
**list.c：**

* **createList：**从标准输入读入数据，创建链表。使用了“虚拟头结点”以简化代码。
* **swapNode：**交换两个结点的数据。
* **printList：**打印链表数据。
* **copyList：**创建一份链表的深拷贝，实现过程类似于创建链表。
* **freeList：**销毁链表，释放所有结点的空间，同样使用了“虚拟头结点”。

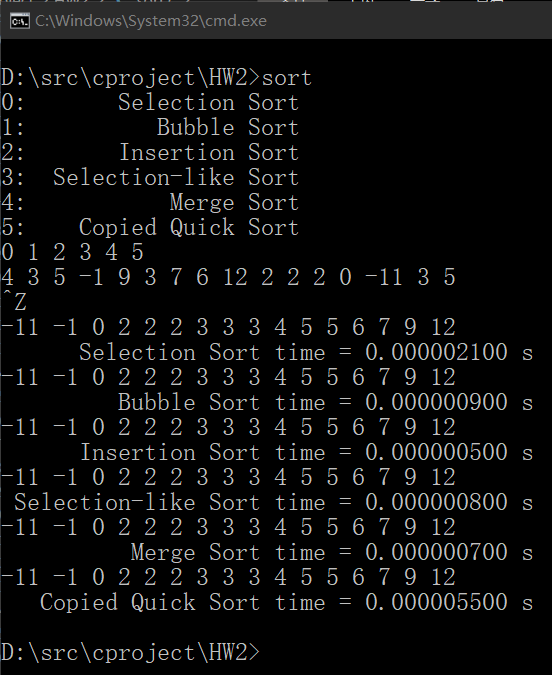
**sort.c：**

* **selectionSort：**选择排序，每次找到最小值与当前元素交换。
* **bubbleSort：**冒泡排序，每次交换相邻逆序元素，将最大值移动到末尾。
* **insertionSort：**插入排序，将当前元素插入到有序区中，直接使用了链表的删除和插入来实现。
* **selectLikeSort：**一种简单的类选择/冒泡排序，比较交换所有逆序对。
* **mergeSort：**归并排序，使用“龟兔赛跑”法找到链表中点（当然直接计数减半也可以），然后分治，用二路归并合并。时间复杂度O(N log N)，空间复杂度O(log N)，是比较容易在链表上实现的O(N log N)算法。
* **copyQSort：**将链表数据复制到数组后调用标准库的qsort，再复制回链表。

**3.3 运行结果展示**

****

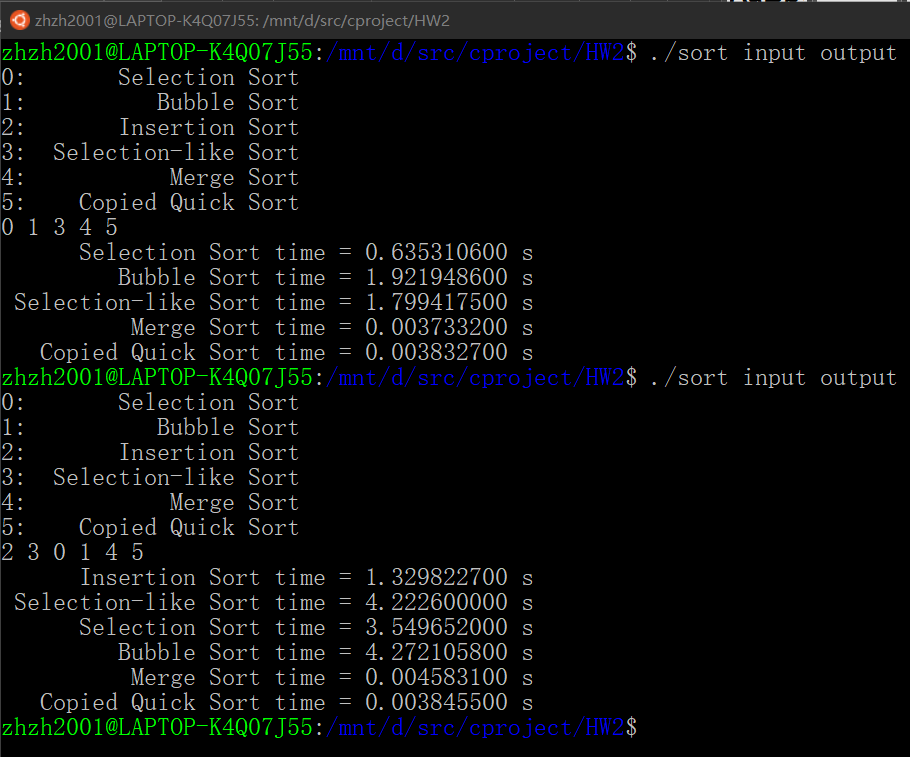
WSL下编译运行，首先选择用于测试的排序算法，然后输入数据，按^D表示EOF，打印各种算法的结果和耗时。

****

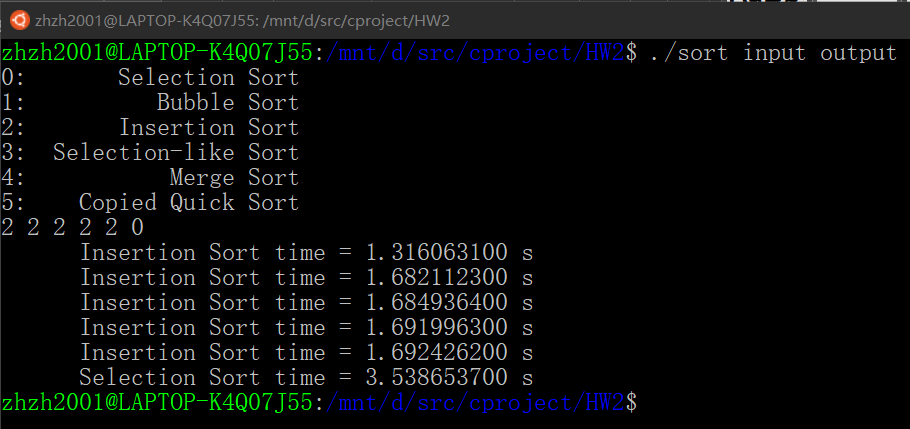
Windows下编译运行，只是用^Z代表EOF。



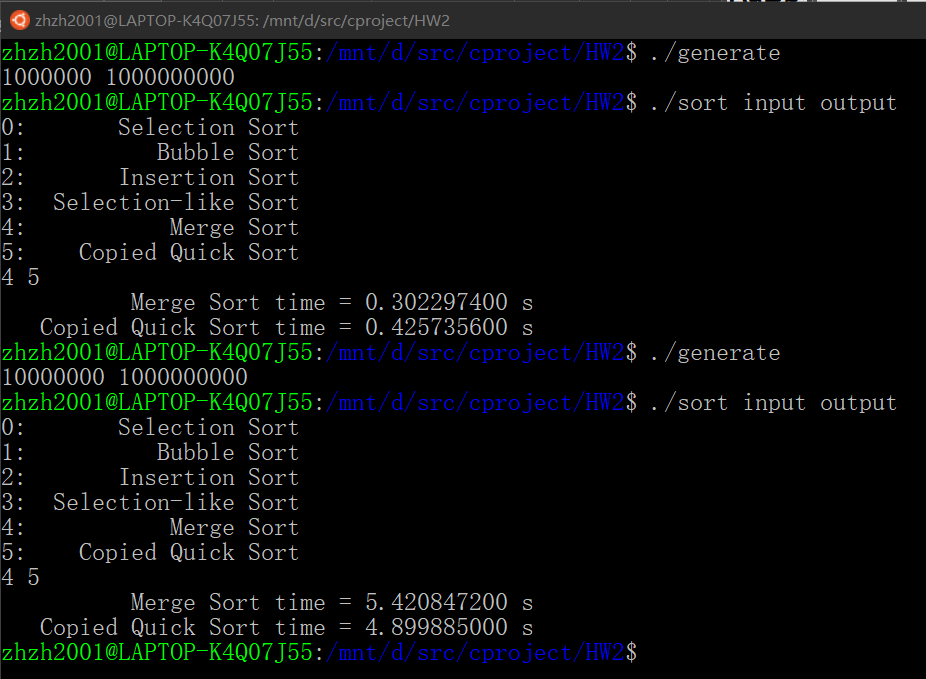
在n=30000规模下的效率比较，看起来实现最简单的类选择排序很慢。然而事实真的是这样吗？通过调整算法运行顺序，发现结果并不是这样。



可见，插入排序后运行的算法明显变慢，推测可能是插入排序复杂的删除插入操作，造成缓存未命中所致，但仍然存在疑问。



然而插入排序多次运行不会减慢自身的速度，但其他算法明显变慢，比较奇怪。



而对比归并排序和复制快速排序，当规模中等时归并排序有一定优势，然而当规模很大时还是快速排序快，推测可能是快速排序的常数，以及数组连续存储优势。