# 《数据结构》课程设计总结

### 学号： 2153299

### 姓名： 杨非凡

### 专业： 计算机科学与技术

### 2023 年 7 月

### 目 录

第一部分 算法实现设计说明 3

1.1题目 3

1.2软件功能 3

1.3设计思想 5

1.4逻辑结构与物理结构 10

1.5开发平台 11

1.6系统的运行结果分析说明 11

1.7操作说明 18

第二部分 综合应用设计说明 22

2.1题目 22

2.2软件功能 22

2.3设计思想 24

2.4逻辑结构与物理结构 31

2.5开发平台 31

2.6系统的运行结果分析说明 32

2.7操作说明 38

第三部分 实践总结 40

3.1.所做的工作 40

3.2.总结与收获 41

第四部分 参考文献 42

### 第一部分 算法实现设计说明

**1.1题目**

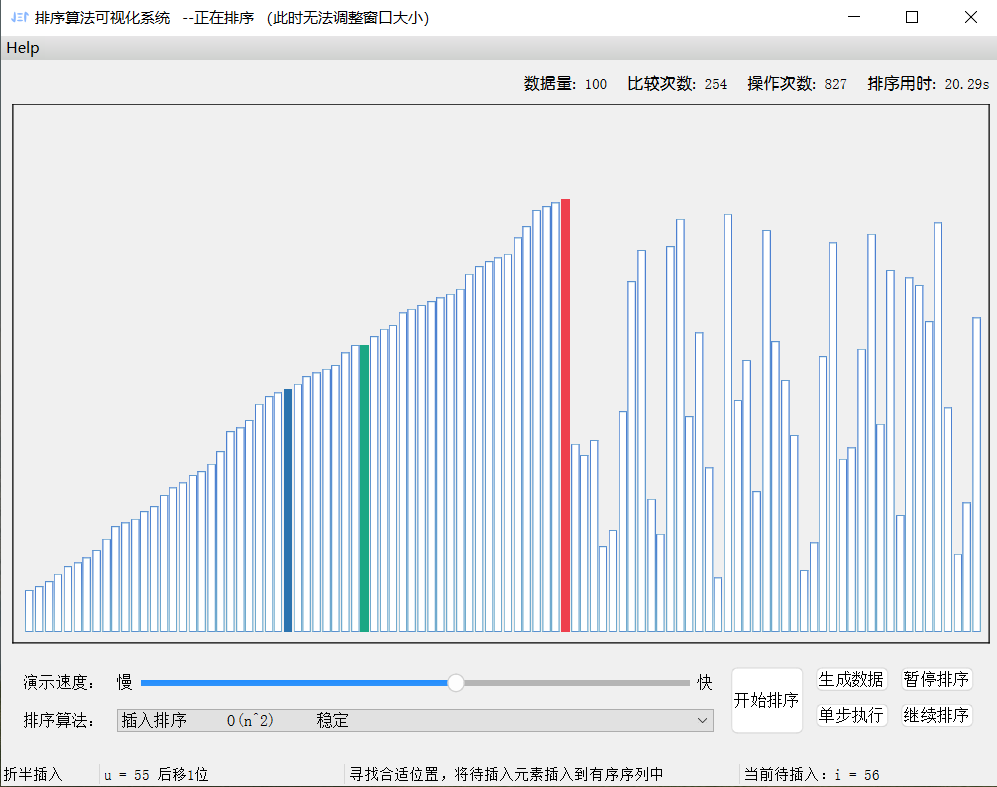
算法实现9：几种排序：要求随机输入一组数据，随时给出某一趟排序的变化情况。

1. 直接插入排序、折半插入排序、希尔排序；
2. 冒泡排序、快速排序；
3. 简单选择排序

**1.2软件功能**

**1.2.1功能设计**

**自顶向下设计**：软件核心功能是对排序算法进行可视化的呈现。对于排序的元素，我们可以采用高度不同的柱状图来表示，这样能够更加直观的体现排序元素的大小关系，也便于演示。软件在一个主界面下需要实现以下功能：排序的控制，能够生成随机的数据，选择排序的方式，同时随时能够暂定排序，并且能调节排序的速度；同时还要对排序使用的时间、比较次数、操作次数以及排序的细节进行呈现。同时还需要设置帮助界面，使用户能够更好地了解如何使用该程序。



上图即为软件界面，软件界面主要由五个部分组成，分别为菜单栏、排序信息展示区（下简称“信息区”）、排序过程演示区（下简称“演示区”）、排序过程控制区（下简称“控制区”）和底部状态栏。

菜单栏：菜单栏中点击Help可以查看帮助，了解如何使用本软件。

信息区：用来展示排序中的数字信息：数据量、比较次数、操作次数、排序用时。

演示区：用来动态展示排序的过程，用到了不同的颜色，通过柱子的高度代表排序数据之间的大小关系，更加直观。白色表示当前步骤中没有动作的元素；蓝色表示当前步骤中被访问的元素，如比较，访问等操作；绿色表示当前步骤中参与写入操作的元素，如交换、更新等操作；红色用于指示当前步骤中的基准值，或用于区分不同元素以便于更加直观的看排序过程。

控制区：用于进行排序进程相关的控制，可以生成随机的数据，选择排序方式，开始排序，一步一步地进行排序，暂停排序，继续排序，以及实时更改排序的速度。

状态栏：状态栏在光标选中某些按钮时会显示该按钮的介绍，同时在排序过程中状态栏还会对当前排序进行到的步骤进行显示,比如(当前正在比较哪两个元素)。

**1.2.2实现方式**

使用Qt5来进行实现，主要采用了Qt5中的信号与槽函数机制，从而实现了内部存储与外观界面的同步，实现时主要遵循了**分模块实现**和**自底向上实现**。

**分模块实现**：程序可以分为如干个模块，比如主窗口模块、排序算法模块、帮助界面模块等。每个模块的界面可以使用Qt中的.ui文件绘制，同时使用qss文件来设定界面的样式；内部逻辑上，将各种排序函数封装成类，通过类的成员函数来实现几种排序方式，同时也将button与槽函数进行connect，确保内部存储的变化与UI的呈现同步。

**自底向上实现**：具体实现时，先定义每个类的属性和相应函数，然后根据定义，设计相应算法自底向上进行实现，逐个击破，最终完成所有程序的设计。

**1.3设计思想**

**1.3.1实现思路**

首先是学习Qt的语言，之前没有过多的接触过有关图形界面的编程，我从零开始学习Qt开发框架，学习了Qt的一些库函数的使用功能，UI界面的设计、信号和槽函数的机制、qss文件的使用等，了解了使用Qt框架开发的基本步骤。

之后就是考虑本次的问题，排序算法其实在本学期初就已经接触过并且完成过相应的作业，比较熟悉了，关键是在于可视化的部分。设计的时候自顶向下考虑，先预先想好需要几个界面，界面的大致样式，每个界面如何分区，绘制好UI文件，并将程序划分为几个类；再自底向上实现，分别实现每个类的功能和每个区域内部的连接，将其与前端界面的部分通过信号与槽函数机制连接起来，就能做到前后端的同步，之后再通过不同部分拼接好整个程序。

具体类的划分如下：

①**Class MainWindow** 负责程序主界面的显示

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

const int HLAYOUTWIDTH = 780; // 默认柱状图区域大小 用于计算缩放倍数

const int HLAYOUTHEIGHT = 410; // 默认柱状图区域大小 用于计算缩放倍数

double multiWidth = 1.0; // 目前窗口宽度的缩放倍数 (窗口默认大小800x600)

double multiHigh = 1.0; // 目前窗口高度的缩放倍数 (窗口默认大小800x600)

double columnWidth;

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

// 按钮

void **on\_btn\_dataGenerate\_clicked**(); // 生成数据按钮

void **on\_btn\_sortingStart\_clicked**(); // 开始排序按钮

void **on\_btn\_sortingPause\_clicked**(); // 暂停排序按钮

void **on\_btn\_sortingContinue\_clicked**(); // 继续排序按钮

void **on\_btn\_oneStep\_clicked**(); // 单步执行按钮

//菜单栏

void **menuAct\_actHelper**();

// 其他

void **setSpeed**(int speed); // 根据滑块设置速度

void **sortingAlgorithm**(int index); // 用户选择了排序算法

void **updateColumns**(int a, int b, int \*newHeights); // 接受排序过程对象发来的交换信号，更新界面中两个柱子的值

void **updateOneColumn**(int c, bool resetPrev, int \*newHeights); // 更新界面中一个柱子的值

void **highLightColumns**(int a, int b); // 单次排序时，高亮正在比较的元素

void **highLightBase**(int base); // 单次排序时，高亮基准值（部分算法）

void **showFinish**(); // 接受排序完成信号，执行完成动画

void **updateSortingTime**(); // 更新排序用时

void **sbShow**(int num, QString text); // 状态栏信息显示 第num个label，显示text内容

private:

void **initMenuBar**(); // 初始化菜单栏和对应的槽

void **dataGenerator**(); // 生成排序数据打乱后显示到界面

void **addDataColumns**(); // 将数据用柱状显示到界面

void **deleteColumns**(); // 删除排序视图柱子

virtual void ***mouseMoveEvent***(QMouseEvent \*event) override; // 按下鼠标左键移动时输出鼠标坐标 用于调试

virtual void ***resizeEvent***(QResizeEvent\* event) override; // 重载窗口大小调整事件，修改柱状图到合适高度

void **initVar**(); // 初始化统计信息变量

void **initLabel**(); // 初始化统计信息显示标签

void **debug\_outNumber**(); // 输出numbers数组查看数据

// 对象

Ui::MainWindow \*ui;

sortAlgorithms \*sortCtrl; // 排序过程总控制对象

QLabel \*columns; // 与numbers数组对应的柱子对象

QTimer\* timer; // 排序计时器

QLabel \*sortName; // 状态里显示算法名称 从左到右第1个label

QLabel \*oprtContent; // 状态栏显示操作内容 从左到右第2个label

QLabel \*sortDscrb; // 状态栏显示算法描述 从左到右第3个label

QLabel \*keyValue; // 状态栏显示关键变量的值 从左到右第4个label

// 菜单栏

QMenu \*menu\_help;

QAction \*actHelper;

// 变量

bool sorted = 0; // 当前数据是否已排序

bool dataGenerating = 0; // 是否正在生成数据

bool datahaved = 0; // 是否已经有了数据

int algorithmSelected = 1; // 当前用户选择的排序算法序号 初始状态为快排\_手工

int sortSpeed = 50; // 用户选择的排序速度

int prevDataNum = 0; // 上次排序演示数据量,用于生成数据中记忆上次数值

int dataNum; // 待排序演示数据量，初值100

int \*numbers; // 整数数组，存储待排序数据

float sortingTime = 0.00; // 排序用时

int prevUpdtA = -1, prevUpdtB = -1; // 上次更新的柱子，记录后用于修改柱子颜色

std::vector<int> prevUpdtC; // 更新单个值时可选择是否重置之前的更新颜色，用于储存哪些值没有重置

int prevCmpA = -1, prevCmpB = -1; // 上次比较时高亮的元素

int prevBase = -1; // 上次的基准值

};

②**class sortAlgorithms**负责排序算法部分

class **sortAlgorithms** : public QThread

{

Q\_OBJECT

public:

**sortAlgorithms**();

~***sortAlgorithms***();

void **setSortingState**(bool state); // 设置排序状态

void **setOneStepState**(bool state); // 设置是否单步排序

bool **isSorting**(); // 获取是否正在排序

bool **isPause**(); // 获取是否处于暂停状态

bool **isOneStep**(); // 获取是否处于单步执行状态

void **setSpeed**(int speed); // 设置排序演示速度

void **setAttribute**(int dataNum, int \*numbers, int algorithmSelected, int speed); // 设置排序属性

int **getCmpCnt**(); // 获取比较次数

int **getOprtCnt**(); // 获取交换次数

void **stopSorting**(); // 立即停止排序用于排序过程中点击重新生成数据

void **pauseSorting**(); // 暂停排序

void **resumeSorting**(); // 继续排序

protected:

void ***run***(); // 重写父类函数，protected保证可以被父类的start调用

void **process**();

signals:

void **sortFinish**(); // 排序结束

void **swapColumn**(int, int, int\*); // 返回需要可视化交换的柱子序号,第三个参数为排序中间过程的数组指针. 注：此信号时更新柱子的值，并不是真正的交换两个柱子，因此部分更新操作也会发送（共用）此信号！

void **updtColumn**(int, bool, int\*); // 更新一个柱子的值 第二个参数为是否重置上一个更新的值的颜色

void **compareColumn**(int, int); // 高亮当前正在比较的数

void **baseColumn**(int); // 高亮基准值（快排）

void **showSBMsg**(int, QString); // 状态栏信息显示 第几个label，显示什么内容

private:

void **swap**(int a, int b); // 交换a b两个位置的数

void **insertSort**(); // 0 插入排序

void **halfinsertSort**(); // 1 折半插入

void **shellSort**(); // 2 希尔排序

void **bubbleSort**(); // 3 冒泡排序

void **quickSort**(int left, int right); // 4 快速排序

void **selectSort**(); // 5 选择排序

QMutex pauseLock;

bool stop; // 线程状态

bool sortingDone; // 是否完成排序

bool pause; // 是否处于暂停状态

bool oneStep; // 是否单步执行

int algorithmSelected; // 使用的排序算法

int speed; // 排序速度

int dataNum; // 排序数据量

int \*numbers; // 排序数据

int cmpCnt = 0; // 比较次数统计

int oprtCnt = 0; // 排序中交换次数

};

③**class helper** 负责帮助界面的显示

namespace **Ui** {

class **helper**;

}

class **helper** : public QMainWindow

{

Q\_OBJECT

public:

explicit **helper**(QWidget \*parent = nullptr);

~***helper***();

private:

Ui::helper \*ui;

};

**1.3.2算法设计基本流程**

本程序中设计到的算法部分为6种排序方式的算法，比较熟悉，但考虑到本程序中还涉及外观界面的呈现，所以要在原有的排序算法的基础上添加该部分。

除了冒泡排序外，其它排序中都有基准值这一概念，所以需要将基准值元素进行高亮；并且在排序中我们应实时显示访问的元素与更改的元素。在6种排序中插入这些外观显示部分即可。6种排序的实现方式如下：

插入排序（Insertion Sort）：从第二个元素开始，将其与已排序部分逐一比较并插入到正确位置。重复以上步骤，直到整个数组有序。

折半插入排序（Binary Insertion Sort）：与插入排序类似，但使用二分查找来寻找插入位置。通过比较中间元素确定插入位置，然后将元素插入。重复以上步骤，直到整个数组有序。

希尔排序（Shell Sort）：将数据分成若干子数组，对每个子数组进行插入排序。子数组的大小逐渐减小，最终整个数组有序。

冒泡排序（Bubble Sort）：从第一个元素开始，比较相邻元素，将较大（或较小）的元素向后交换。重复以上步骤，每次将一个最大（或最小）的元素冒泡至末尾。重复该过程，直到整个数组有序。

快速排序（Quick Sort）：选择一个枢纽元素，将数组分成两部分，左边的元素小于枢纽，右边的元素大于枢纽。递归地对左右两部分进行快速排序，直到整个数组有序。

选择排序（Selection Sort）：从未排序部分选择最小（或最大）的元素，与已排序部分的末尾交换位置。扩展已排序部分，重复以上步骤，直到整个数组有序。

**1.4逻辑结构与物理结构**

**逻辑结构**：本程序中使用的逻辑结构为线性表，线性表中存储了排序的元素，各个元素之间是线性排列的关系。我们可以通过每个元素在线性表中所占的位置来访问它们的值，排序算法也是基于此完成的，通过交换元素使得线性表中的元素最后有序排列。

**物理结构**：在本程序中使用数组来实现线性表的功能，由于程序中规定了最大数据规模为300（数据过多不利于图形化显示），所以数组的大小开到300即可。在程序中可以通过数组下标作为索引来访问线性表中的元素。

**1.5开发平台**

**开发平台：**

计算机型号 ：Lenovo Legion Y70002021

CPU ：11th Gen Intel® CoreTM i5-11400H @ 2.70GHz

操作系统 ：Windows 10

开发语言 ：C++(C++11标准以上)

开发框架 ：Qt Version 5.14.2

集成开发环境 ：Qt

编译器 ：Qt 5.14.2 (MinGW 7.3.0 32-bit)

**运行环境：**

可在上述集成环境下运行。

同时通过windeployqt.exe及Enigma Virtual Box进行整合压缩为一个sorting\_visulization\_boxed.exe文件，可在普通windows机型（winsows 7以上版本）直接运行。

**开源代码的使用：**

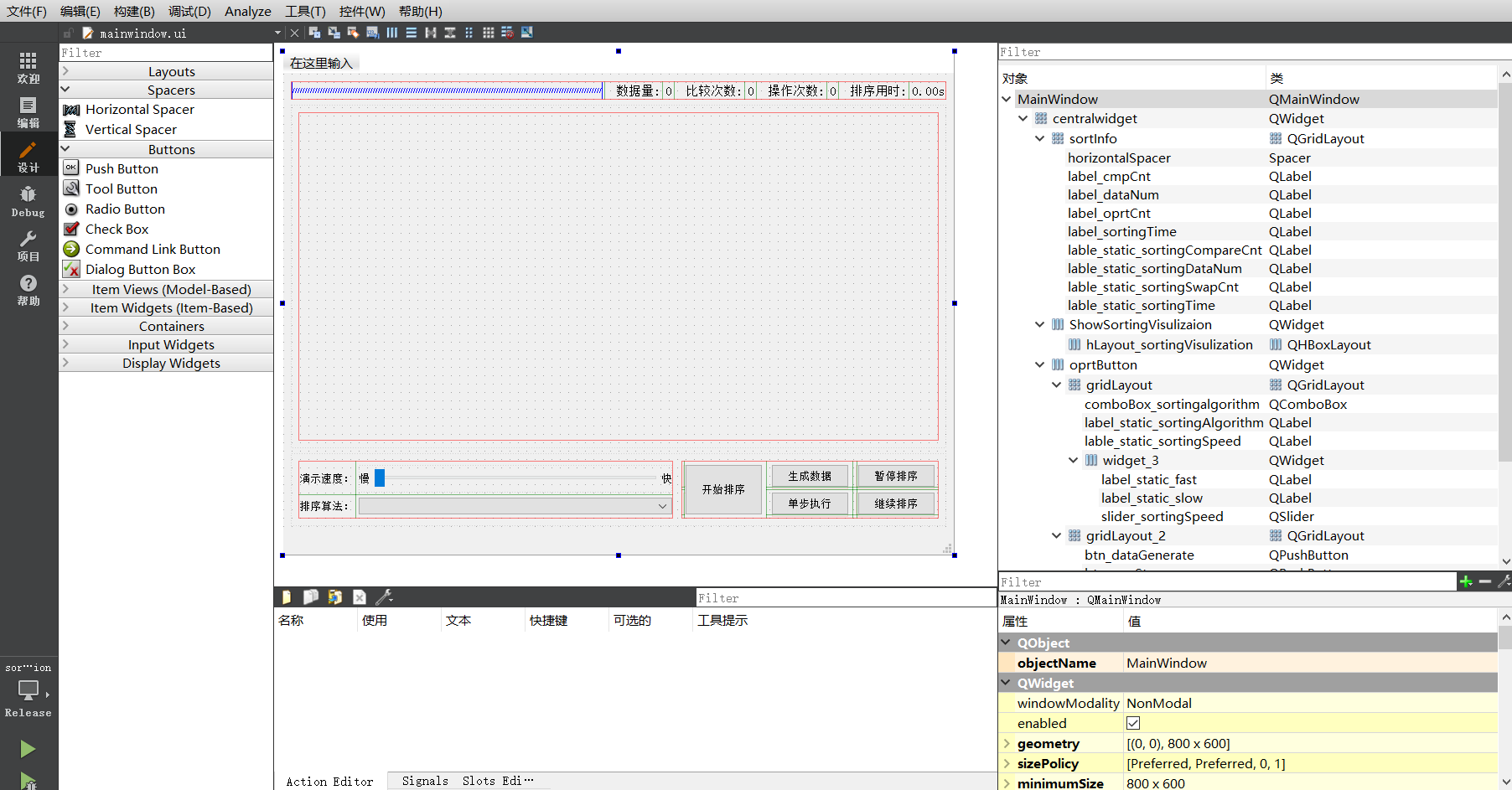
本程序的样式文件style.qss来自：

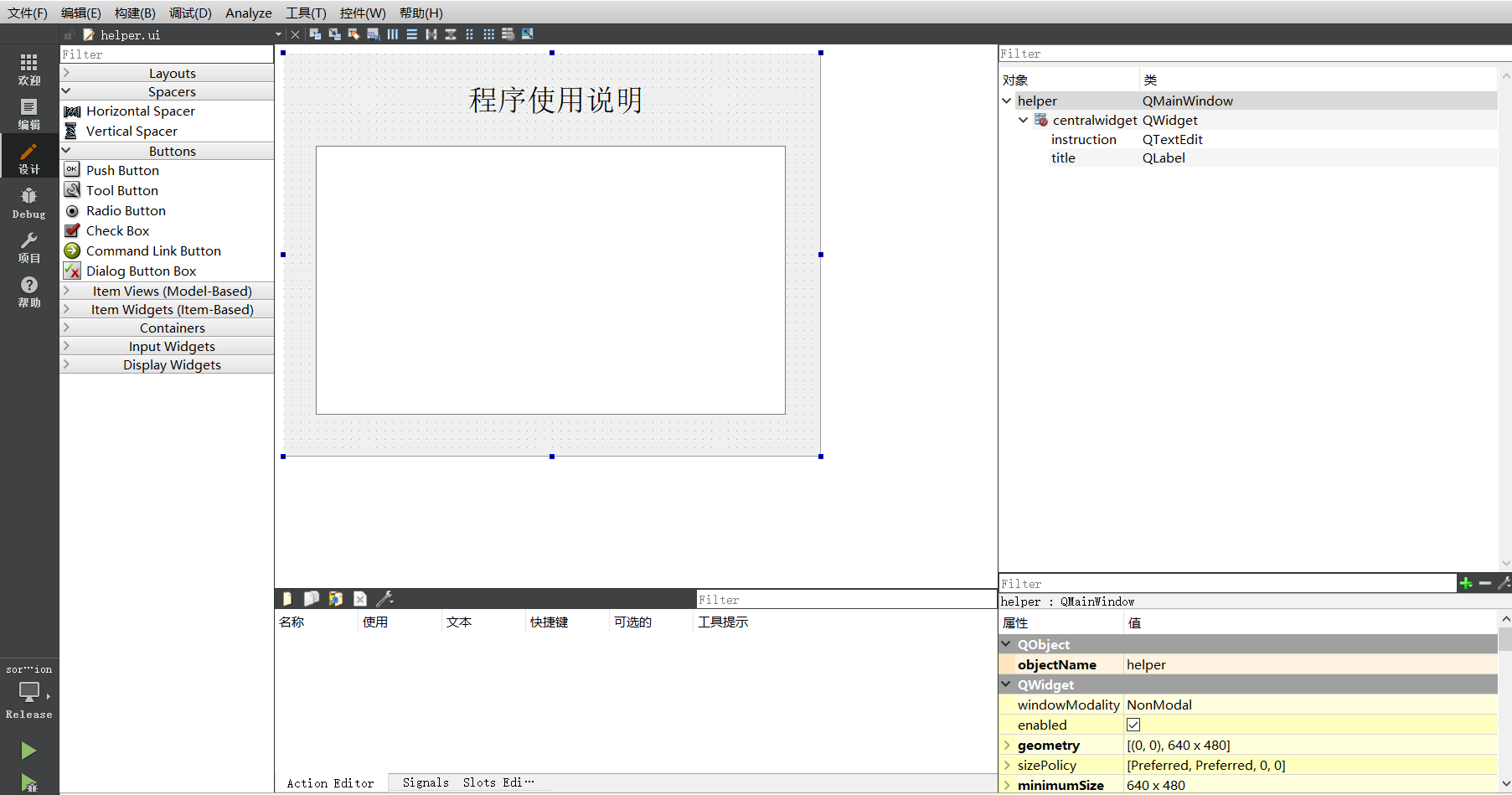
<https://github.com/GTRONICK/QSS/blob/master/Aqua.qss>

**1.6系统的运行结果分析说明**

**1.6.1调试及开发的过程**

**开发过程**中，先按照设计思路将各文件建好，之后先通过.ui文件绘制好界面：

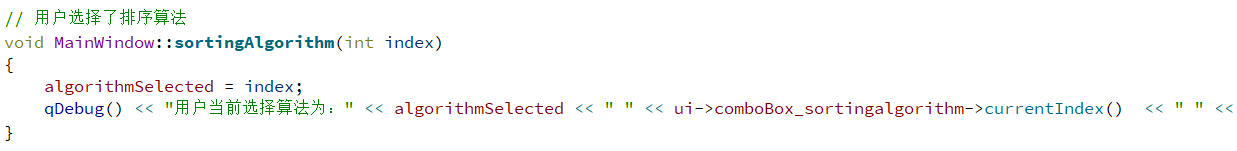




之后再去定义各类中需要用到的私有变量以及成员函数，确定好类与类之间的继承与派生关系。

最后通过信号与槽函数机制，将前端与后端联系起来，实现前后端的同步。同时通过qt中的资源文件来向程序中插入icon图标以及使用qss文件来改变其样式。

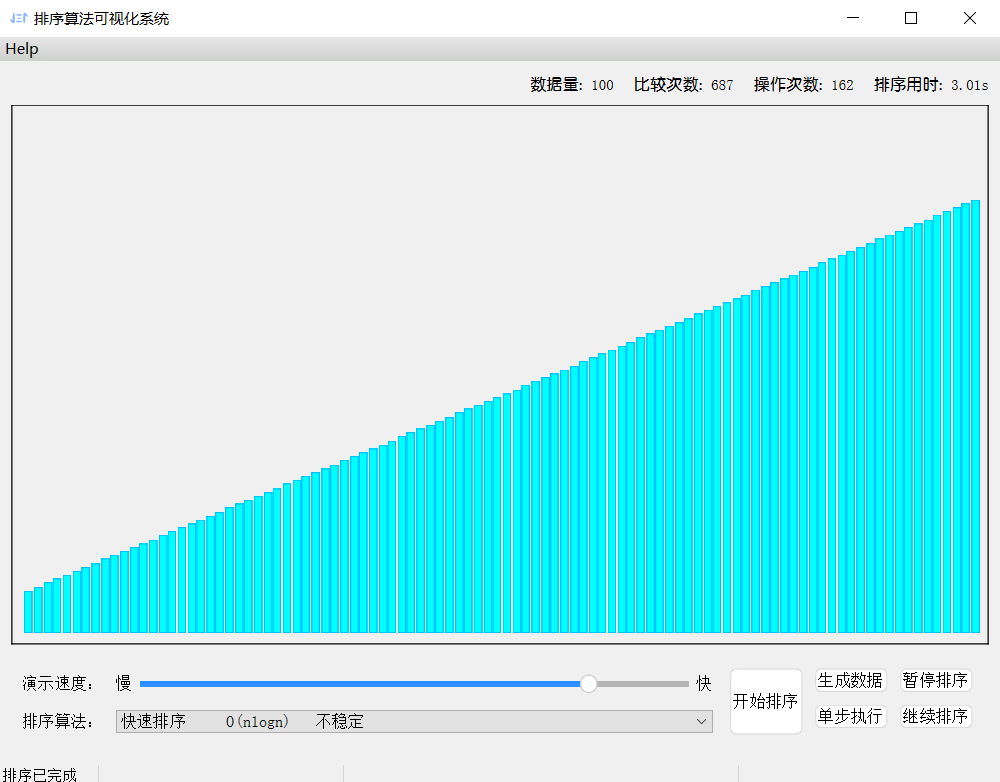
**调试过程**中，主要使用qt中的qdebug对中间变量进行输出以及配合qt中自带的调试功能来进行调试。



**1.6.2所开发软件达到的成果**

**正确性及稳定性**：

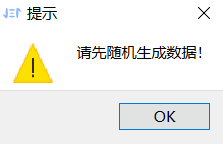
我对于每种排序算法，都随机生成了20次数据，并在不同的排序速度下进行排序，排序都能稳定进行并且最后能够将数据排列为升序。同时在排序的过程中，程序能够稳定运行，暂停排序、继续排序以及单步排序都能够正常使用。最终排序的结果如下。



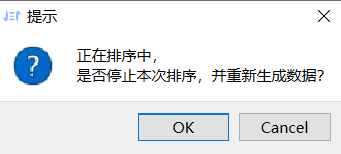
**容错性**：

程序具有较好的容错性，能够极大程度限制许多非法操作，下面通过几个操作进行说明。

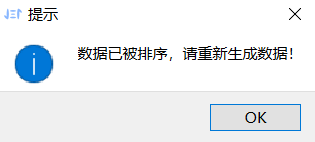
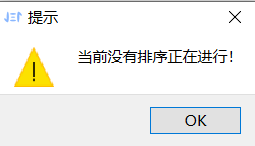
①在没有生成数据的情况下点击开始排序按钮会进行提示：



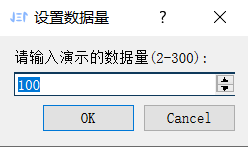
②在排序的过程中，不允许改变排序算法，此时点击生成数据会提示：



③对于已经结束的排序，数据已经有序，此时点击开始排序和暂停排序分别会提示：

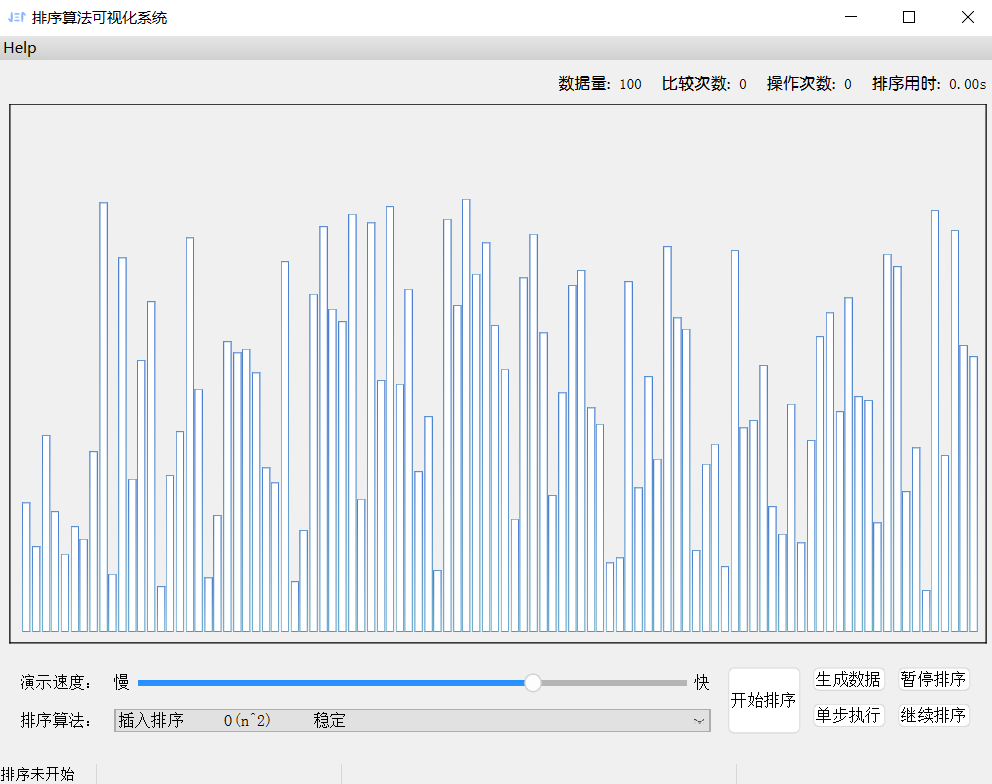
 

④利用控件保证输入的合理性，在设置数据量时，输入不在[2,300]范围内的数无效：

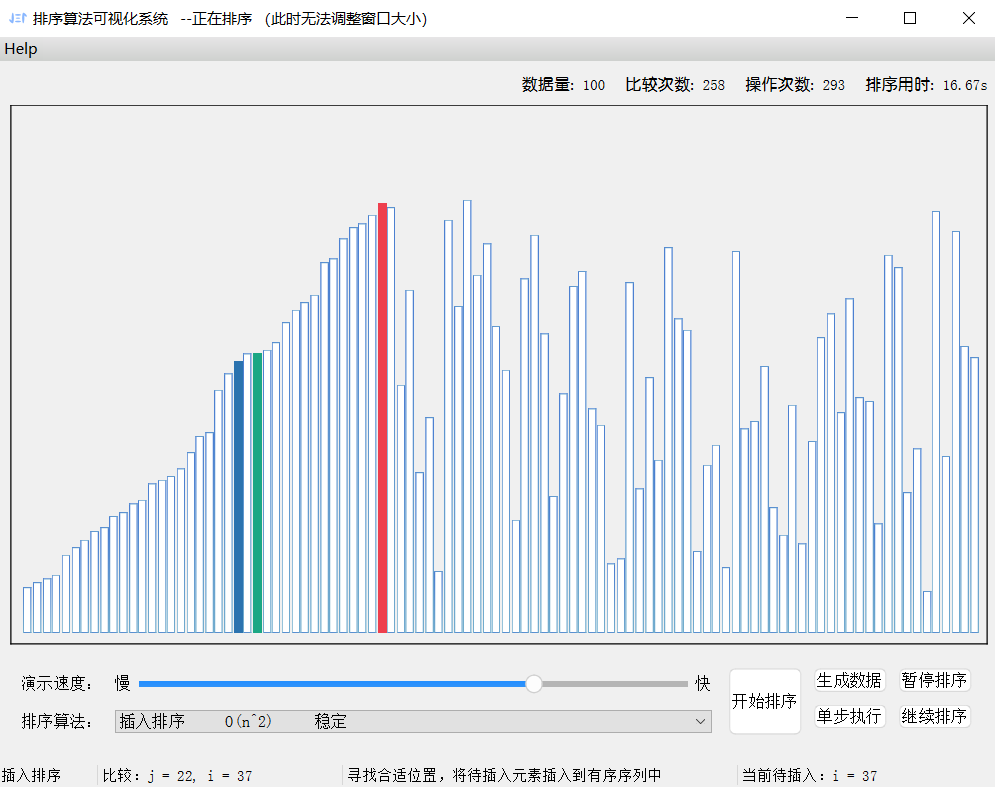


**1.6.3通过案例说明运行结果**

初始数据如图:

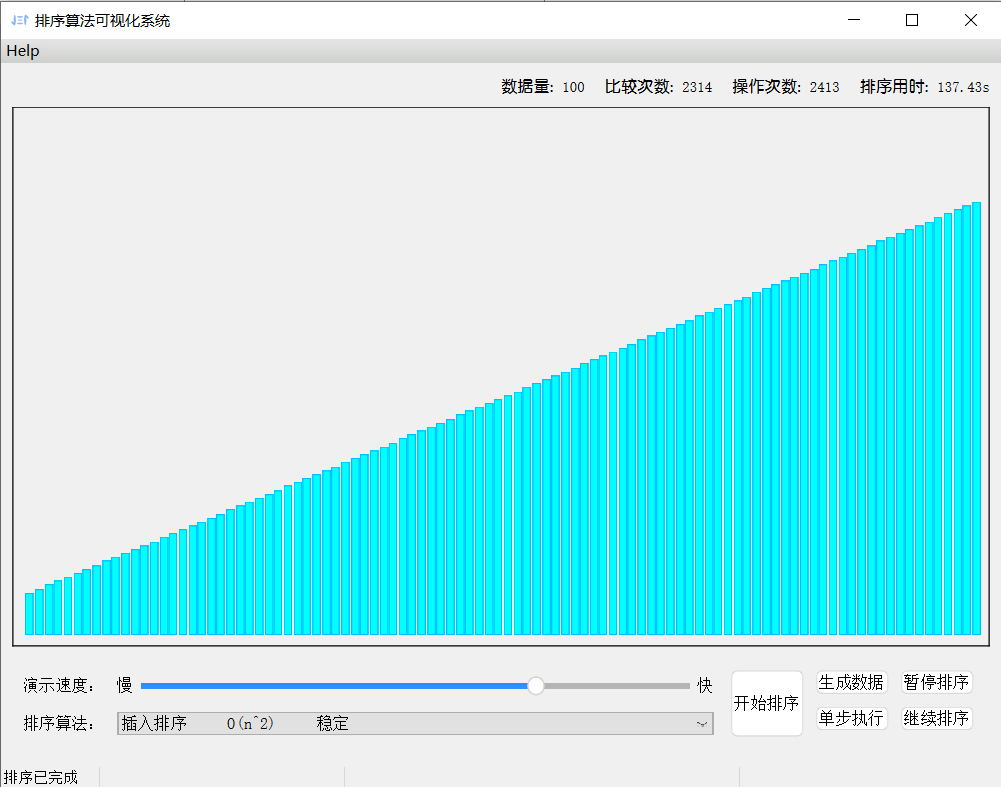


我们选择的排序算法是插入排序（其它排序同理），点击开始排序后，排序正常进行，排序过程中的截图如下：

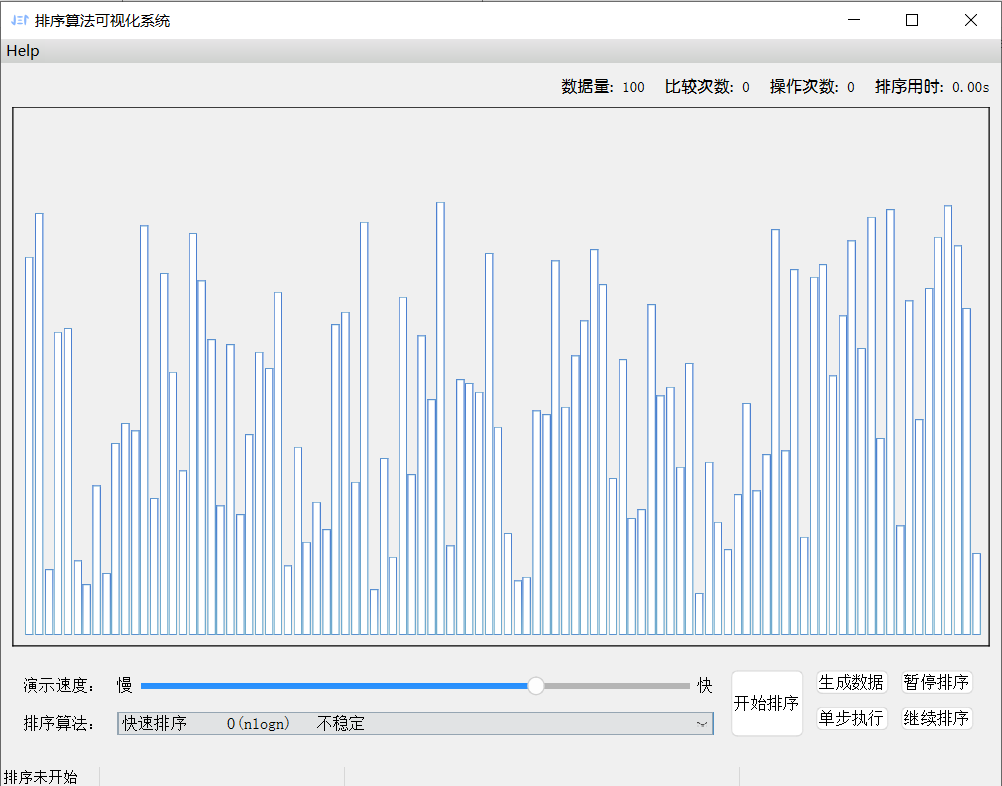


通过界面的底部状态栏，可以知道此时插入排序进行到第37个位置，前面36个元素已经有序，并且此时正在比较第22个元素和插入值的大小关系，第23个元素正在进行修改。通过界面的信息区可知，这次排序的数据量为100，排序到此时的比较次数为258，操作次数为293，所用时间为16.67s。

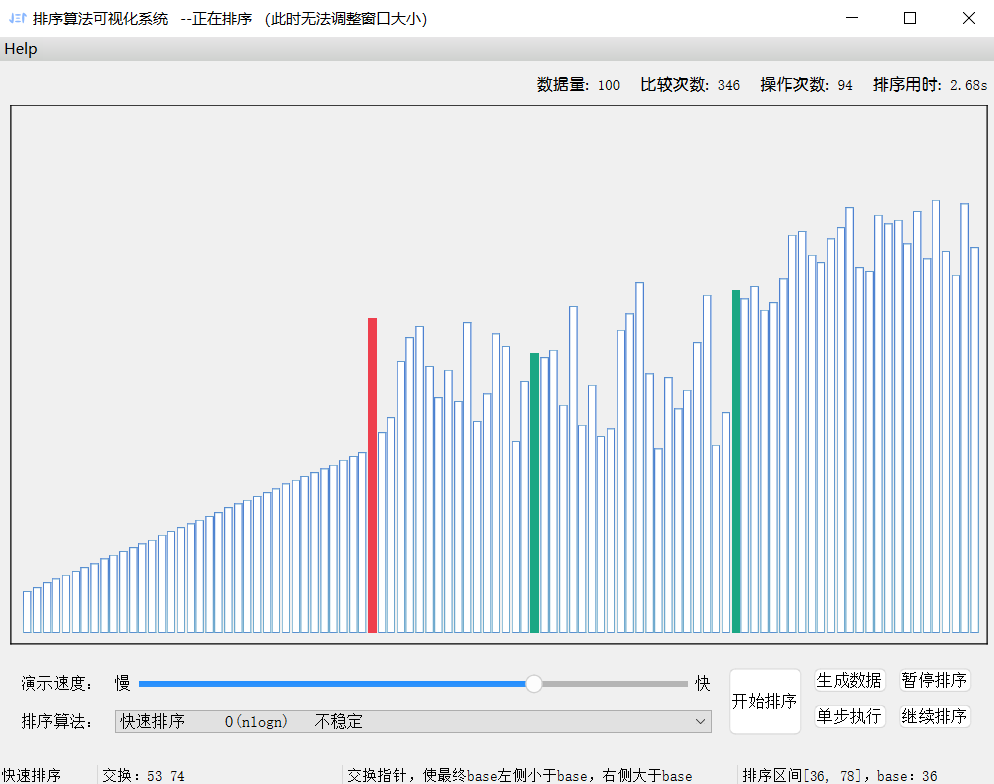
排序完成后界面如下：



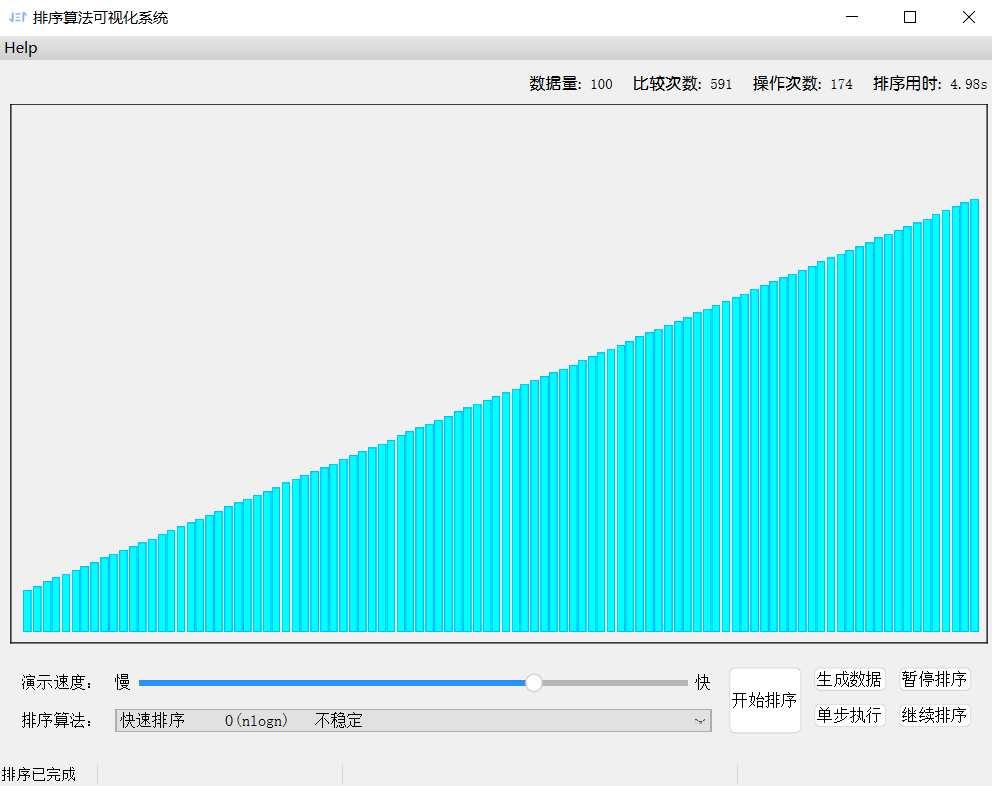
数据已经完全排好序，可见此次排序的总比较次数为2314，总操作次数为2413，用时137.43s。之后我们再次点击生成数据，生成新的数据，同时将排序算法改为时间复杂度更低的快速排序，演示速度不变：



之后点击开始排序，排序过程中的截图如下：



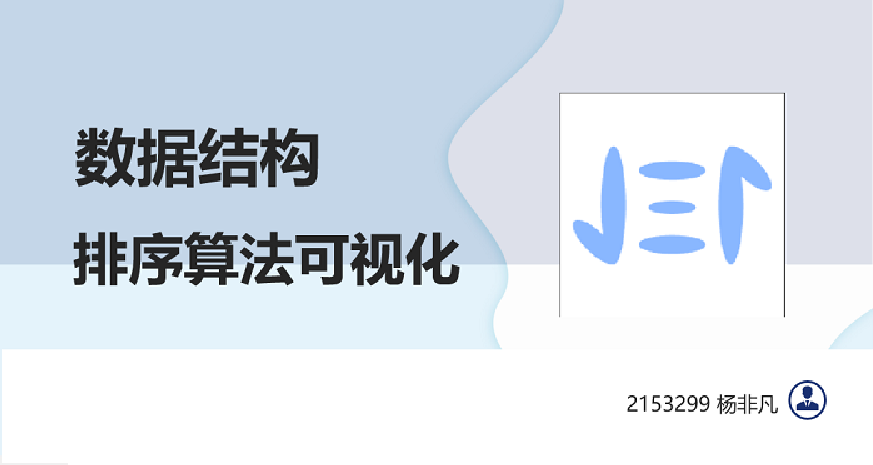
可见此时的排序基准值为第36个元素，排序区间为[36,78]，正在进行交换指针操作，通过界面的信息区可知，这次排序的数据量为100，排序到此时的比较次数为346，操作次数为94，所用时间为2.68s。排序完成后截图如下：



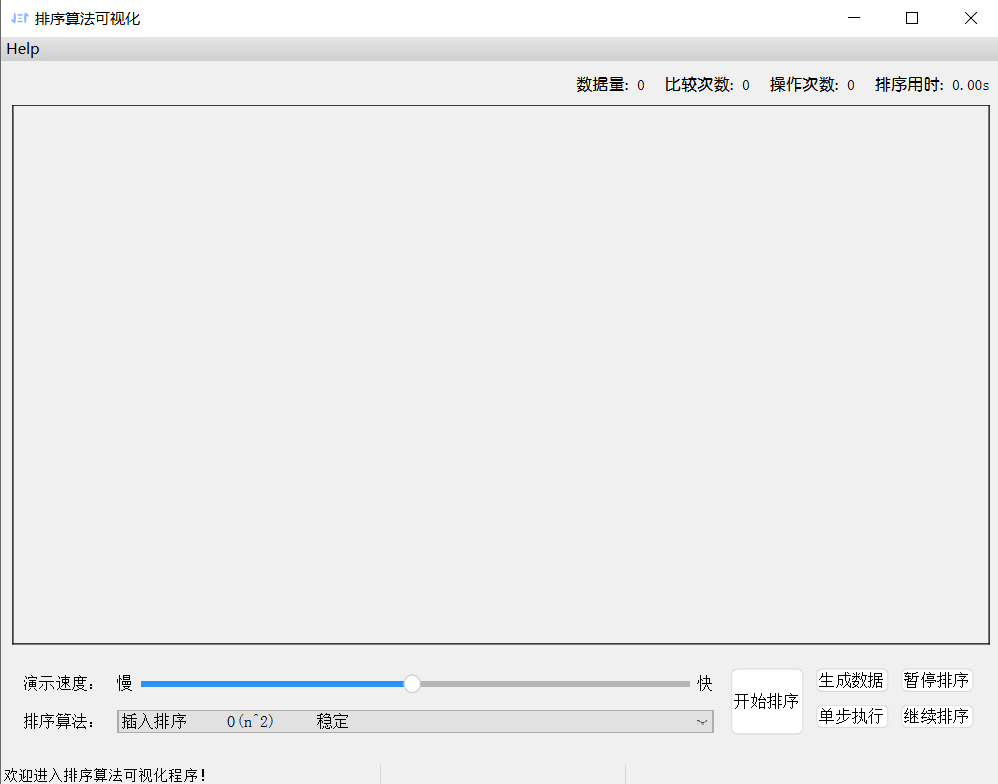
数据已经完全排好序，可见此次排序的总比较次数为591，总操作次数为174，用时4.98s，用时远小于插入排序。

**1.7操作说明**

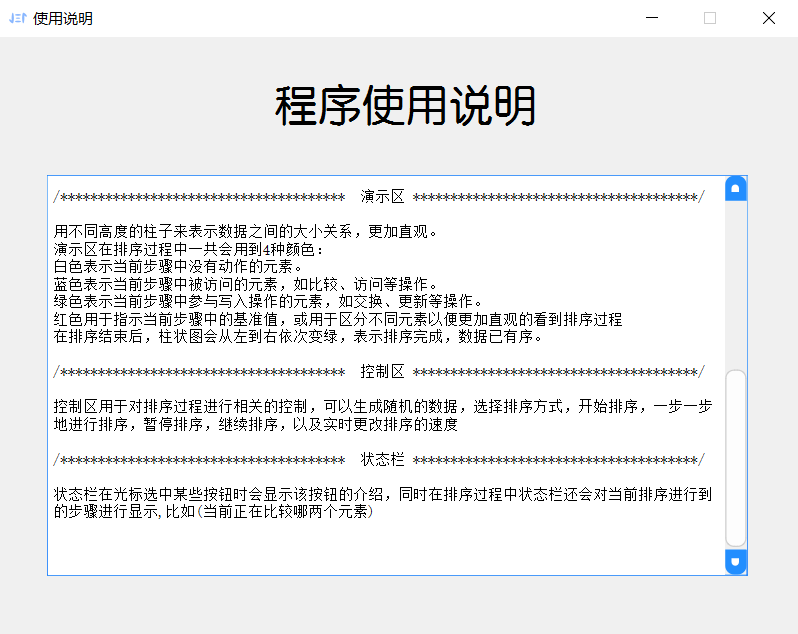
①首先打开程序后会显示如下的初始界面，持续2s左右



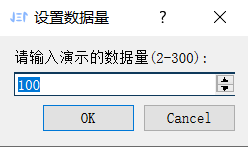
②之后进入如下的主界面：



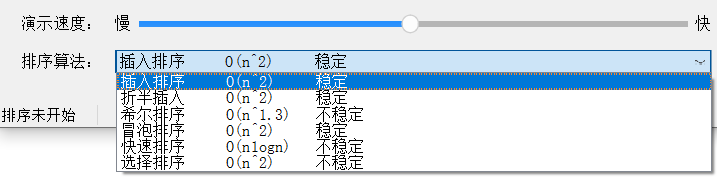
③可以点击导航栏的Help来查看程序的使用说明：



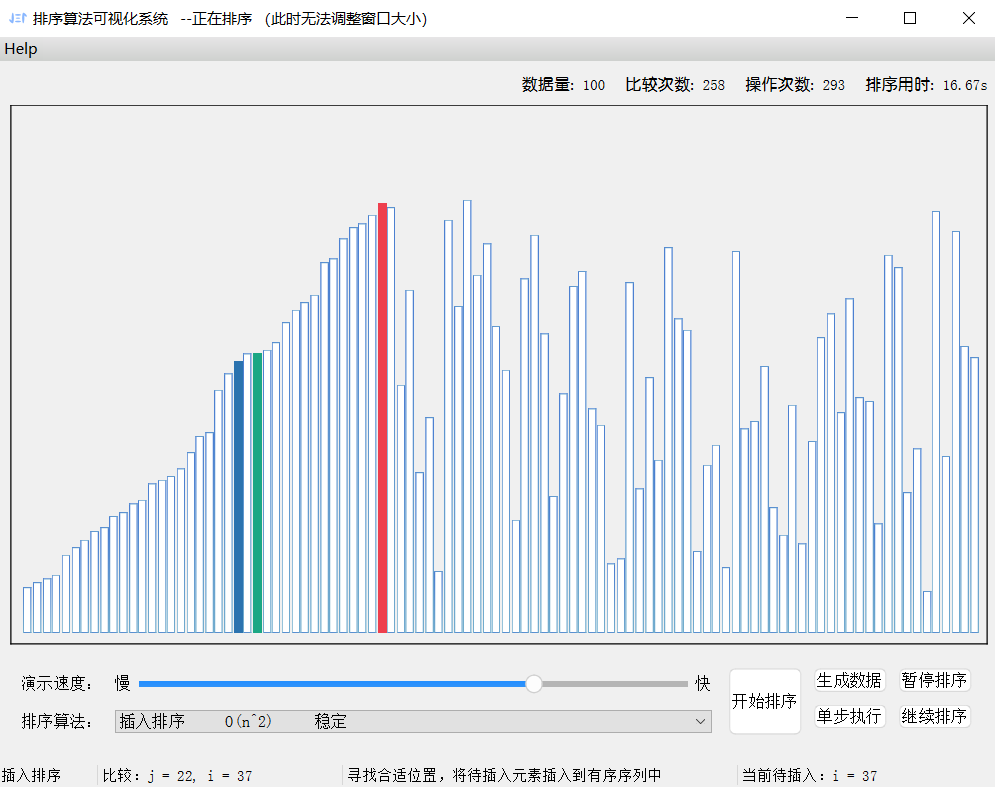
④点击生成数据来选择随机生成数据的规模：



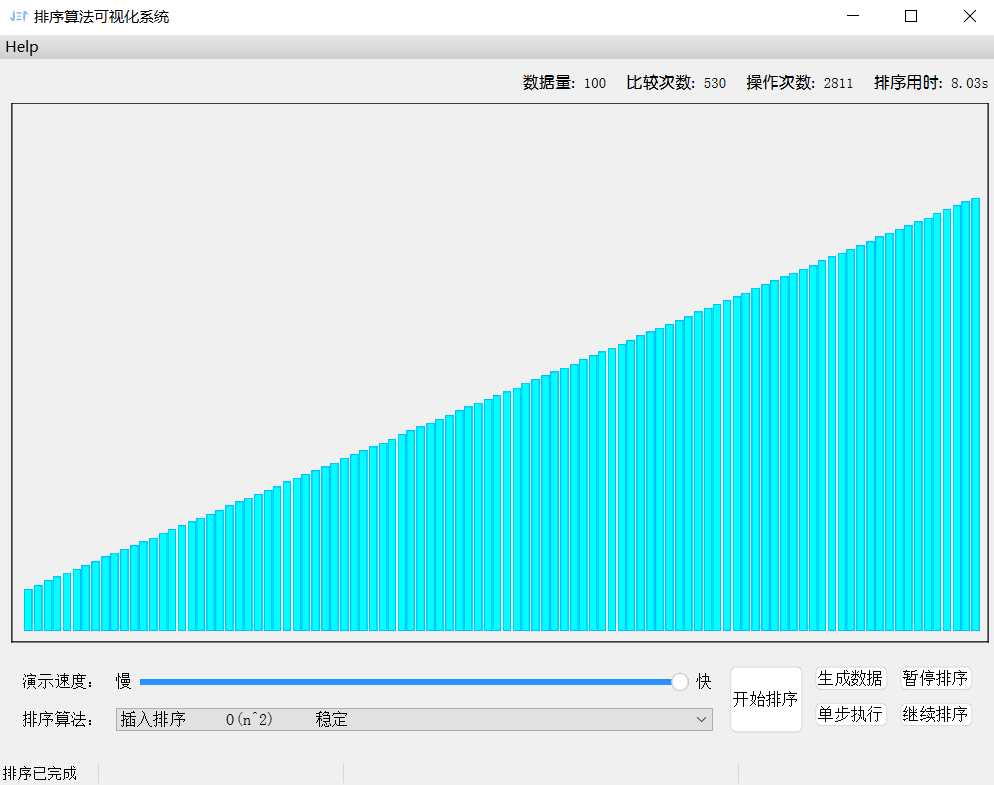
⑤确定之后可以选择需要的排序算法以及排序速度：



⑥之后点击开始排序即可进行排序，排序过程中信息区、演示区、状态栏都会实时变化。



⑦排序完成后状态栏会有提示，并且会展示排序结果，此时点击生成数据即可回到④的状态，可以重新选择排序的算法和速度进行新一轮排序。



### 第二部分 综合应用设计说明

**2.1题目**

（本人学号尾号为9，主动将综合应用更换为难度更高的题目2）。

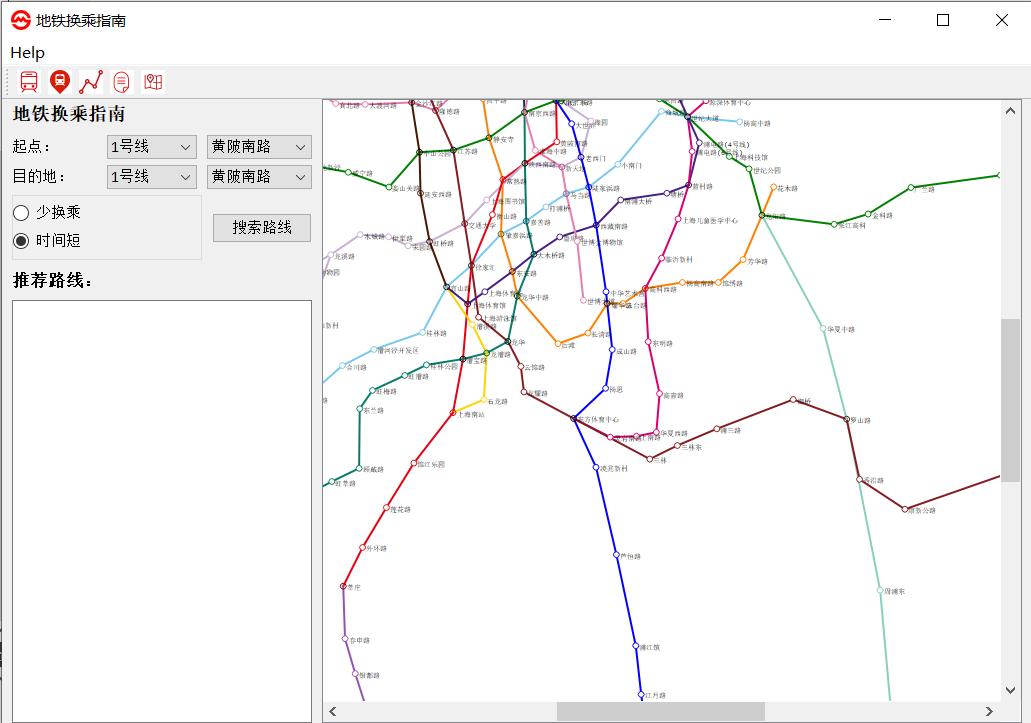
综合应用2：上海的地铁交通网络已经基本成型，建成的地铁线十多条，站点上百个，现需建立一个换乘指南打印系统，通过输入起点站和终点站，打印出地铁换乘指南，指南内容包括起点站、换乘站、终点站。

1. 图形化显示地铁网络结构，能动态添加地铁线路和地铁站点。
2. 根据输入起点站和重点站，显示地铁换乘指南。
3. 通过图形界面显示乘车路径。

**2.2软件功能**

**2.2.1功能设计**

**自顶向下设计**：软件核心功能是设计一套地铁换乘系统。内部可以通过二维数组来存储地铁网络图结构，同时用集合来存储每个站点所在的线路和每个线路包含的站点。软件在一个主界面下需要实现以下功能：读取文件，将上海的地铁线路信息通过图的方式展现出来；同时支持给出起点和终点的情况下的换乘搜索，能够搜索时间最短和换乘最少得线路，并且还要支持动态添加线路和站点等。还要给出程序的使用说明，来帮助用户了解如何学习和使用该程序。



上图即为软件界面，软件界面主要由四个部分组成，分别为菜单栏、动态添加栏、地铁线路展示区（下简称“展示区”）和地体换乘查询区（下简称“查询区”）。

菜单栏：菜单栏中点击Help可以查看帮助，了解如何使用本软件。

动态添加栏：用来进行动态添加线路、站点、站点之间的链接。

展示区：用来展示地铁线路。同时在换乘查询后会将查询结果通过图形界面显示。

查询区：用来进行换乘查询，可以更改起点终点以及搜索换乘路线的偏好来查询相应的路线。

**2.2.2实现方式**

使用Qt5来进行实现，主要采用了Qt5中的信号与槽函数机制，从而实现了内部存储与外观界面的同步，实现时主要遵循了**分模块实现**和**自底向上实现**。

**分模块实现**：程序可以分为如干个模块，比如主窗口模块、地铁换乘模块、动态添加模块、站点模块、帮助界面模块等。每个模块的界面可以使用Qt中的.ui文件绘制，内部逻辑上，将各模块封装成类进行实现。

**自底向上实现**：具体实现时，先定义每个类的属性和相应函数，然后根据定义，设计相应算法自底向上进行实现，逐个击破，最后再完成给程序添加icon图标等工作，完成整个程序。

**2.3设计思想**

**2.3.1实现思路**

考虑本次的问题，关键的算法部分在与最少时间和最短换乘的搜索，最短时间我们可以贪心考虑，使用优先队列优化的Dijstra算法进行求解，最少换乘使用广度优先搜索即可，剩下的部分在于地铁网络图的呈现以及前端的展示。设计的时候自顶向下考虑，先预先想好需要几个界面，界面的大致样式，每个界面如何分区，绘制好UI文件，并将程序划分为几个类；再自底向上实现，分别实现每个类的功能和每个区域内部的连接，将其与前端界面的部分通过信号与槽函数机制连接起来，就能做到前后端的同步，之后再通过不同部分拼接好整个程序。

具体类的划分如下：

①**Class MainWindow** 负责程序主界面的显示

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

explicit **MainWindow**(QWidget \*parent = 0);

~***MainWindow***();

public slots:

//换乘查询槽函数

void **updateTranserQueryInfo**();

void **transferStartLineChanged**(QString lineName);

void **transferDstLineChanged**(QString lineNames);

void **transferQuery**();

//6个顶部按钮的槽函数

void **on\_actionAddLine\_triggered**();

void **on\_actionAddStation\_triggered**();

void **on\_actionAddConnect\_triggered**();

void **on\_actionAddByText\_triggered**();

void **on\_actionLineMap\_triggered**();

void **on\_actionuseHelp\_triggered**();

//添加列表视图部件变化槽函数

void **tabWidgetCurrentChanged**(int index);

//动态添加槽函数

void **addLine**();

void **addStation**();

void **addConnection**();

void **addByText**();

protected:

Ui::MainWindow \*ui; //主窗口UI

Graphics\_view\_zoom \*myView; //自定义视图，用于鼠标缩放

QGraphicsScene \*scene; //场景

SubwayGraph\* subwayGraph; //后端管理类

ManageLines\* manageLines; //添加功能前端管理类

AppHelp\* appHelp; //帮助显示类

//绘制网络图的边

void **drawEdges** (const QList<Edge>& edgesList);

//绘制网络图的站点节点

void **drawStations** (const QList<int>& stationsList);

//由线路表计算混合颜色

QColor **getLinesColor**(const QList<int>& linesList);

//获得线路表的名字集

QString **getLinesName**(const QList<int>& linesList);

//将站点的经纬度地理坐标转为视图坐标

QPointF **transferCoord**(QPointF coord);

private:

void **MainConnect**();

};

②**class help**负责帮助界面的呈现

class **AppHelp** : public QWidget

{

Q\_OBJECT

public:

explicit **AppHelp**(QWidget \*parent = 0);

void **CreateHelpWindow**();

private slots:

void **listWidgetChanged**(QListWidgetItem\*);

private:

QListWidget \*help\_left;

QTextBrowser \*html\_content;

};

③**class managelines**负责动态添加线路部分

namespace **Ui** {

class **ManageLines**;

}

class **ManageLines** : public QDialog

{

Q\_OBJECT

private slots:

void **on\_doubleSpinBoxLatitude\_valueChanged**(double x);

void **on\_doubleSpinBoxLongitude\_valueChanged**(double x);

void **on\_listWidget\_itemClicked**(QListWidgetItem \*item);

void **on\_lineEditLineName\_textChanged**(const QString &x);

void **on\_pushButtonChooseColor\_clicked**();

void **on\_lineEditStationName\_textChanged**(const QString &x);

public:

explicit **ManageLines**(QWidget \*parent = 0);

~***ManageLines***();

//设置不同部件可见

void **setAddLineVisible**();

void **setAddStationVisible**();

void **setAddConnectionVisible**();

void **setAddByTextVisible**();

void **updateLinesListWidget**();

void **updateComboBox**();

protected:

Ui::ManageLines \*ui;

QVector<QWidget\*> tabWigetsVector; //保存tab部件指针

QVector<QIcon> tabIconVector; //保存tab部件Icon

QVector<QString> tabTextVector; //保存tab部件标题

QList<QString> linesNameList,linesSelected,stationsNameList;

QString lineName,stationName;

QColor lineColor;

double longitude,latitude;

//声明友元

friend class MainWindow;

};

④class mouseviewzoom复杂鼠标滚轮缩放界面

⑤class station复杂地铁站点的信息存储

class **Station**

{

protected:

int id;

QString name;

double longitude, latitude;

QSet<int> linesInfo;

//所有站点的边界位置

static double minLongitude, minLatitude, maxLongitude, maxLatitude;

public:

**Station**();

**Station**(QString nameStr, double longi, double lati, QList<int> linesList);

protected:

int **GetDistance**(Station other);

friend class SubwayGraph;

friend class QTextStream;

};

⑥class subwaygraph负责地铁图的存储与换乘查询

//定义边类型

typedef QPair<int,int> Edge;

class **SubwayGraph**;

class **QTextStream**;

//线路类

class **Line**

{

protected:

//储存线路相关信息

int id;

QString name;

QColor color;

QVector <QString> fromTo;

QSet<int> stationsSet;

QSet<Edge> edges;

public:

//构造函数

**Line**(){};

**Line**(QString lineName, QColor lineColor):name(lineName), color(lineColor)

{};

//声明友元

friend class SubwayGraph;

friend class QTextStream;

};

class **Node**{

public:

int stationID; //邻接点ID

double distance; //两点距离

//构造函数

**Node**(){};

**Node**(int s, double dist) :stationID(s), distance(dist){};

bool operator > (const Node& n) const

{

return this->distance>n.distance;

}

};

//后端管理类

class **SubwayGraph**

{

protected:

QVector<Station> stations; //存储所有站点

QVector<Line> lines; //存储所有线路

QHash<QString, int> stationsHash; //站点名到存储位置的hash

QHash<QString, int> linesHash; //线路名到存储位置的hash

QSet<Edge> edges; //所有边的集合

QVector<QVector<Node>> graph; //地铁线路网络图

public:

//构造函数

**SubwayGraph**();

QString **getLineName**(int l);

QColor **getLineColor**(int l);

int **getLineHash**(QString lineName);

QList<int> **getLinesHash**(QList<QString> linesList);

QList<QString> **getLinesNameList**();

QList<QString> **getLineStationsList**(int l);

QList<int> **getStationLinesInfo**(int s);

QList<int> **getCommonLines**(int s1, int s2);

int **getStationHash**(QString stationName);

QList<QString> **getStationsNameList**();

QString **getStationName**(int s);

QPointF **getStationCoord**(int s);

QPointF **getMinCoord**();

QPointF **getMaxCoord**();

//添加新线路

void **addLine**(QString lineName, QColor color);

void **addStation**(Station s);

void **addConnection**(int s1, int s2, int l);

//获取网络结构，用于前端显示

void **getGraph**(QList<int>&stationsList, QList<Edge>&edgesList);

bool **queryTransferMinTime**(int x, int y,QList<int>&stationsList,QList<Edge>&edgesList);

bool **queryTransferMinTransfer**(int x, int y,QList<int>&stationsList,QList<Edge>&edgesList);

bool **readFileData**(QString fileName);

private:

//清空数据

void **clearData**();

//插入一条边

bool **insertEdge**(int s1, int s2);

//更新边界经纬度

void **updateMinMaxLongiLati**();

//生成图结构

void **makeGraph**();

};

**2.3.2数据结构的设计思想及选择理由**

程序最后选用了线性表和集合这两种数据结构，二维的线性表可以存储地铁线路图，用来表示每两个站点之间变的情况；集合能够维护每个站点属于哪些线路以及每个线路上有哪些站点，为bfs和dijstra算法做好铺垫。

选择这两种数据结构的理由：比较简单并且对于本题来说很合适，这两种数据结构应用起来较为简单并且能够满足本题目的需求。同时这两种数据结构在后续动态添加上都很容易，二维线性表的话直接直接扩展即可，集合的话也可以加入新的元素，能够满足本题种动态添加部分的需求。

**2.3.3算法设计基本流程**

本程序中设计到的算法部分为图中的搜索问题，比较熟悉，但考虑到本程序中还涉及外观界面的呈现，所以只要在搜索的过程中将临时结果储存起来，最后搜索完成后进行输出即可。

最使用广度优先搜索（Breadth-First Search，BFS）算法来查找地铁站点之间的最短换乘路径。广度优先搜索从起始站点开始，逐层扩展搜索，直到找到目标站点或确定无法到达。该算法使用队列数据结构来维护待访问的站点，以确保按层级顺序进行搜索。

使用Dijkstra算法来查找地铁站点之间的最短时间路径。Dijkstra算法是一种贪心算法，它从起始站点开始，通过不断选择当前距离最短的节点来更新最短距离和前驱节点，直到找到目标站点或确定无法到达。该算法使用优先队列（priority\_queue）来维护候选节点，并按照最短距离的顺序进行访问。

**2.4逻辑结构与物理结构**

**逻辑结构**：

线性表（Linear List）：用于存储地铁线路图的二维线性表（可能是二维数组或其他数据结构）。这个线性表用来表示每两个站点之间的变化情况，它在逻辑上描述了地铁线路的连接和站点之间的关系。

集合（Set）：用于维护每个站点属于哪些线路以及每个线路上有哪些站点的集合。这个集合在逻辑上描述了站点与线路之间的关联关系。

**物理结构**：

线性表的物理结构：线性表的物理结构是qt中的二维Qvector，其中行列表示站点，每个元素表示这两个站点间的线路关系。这个物理结构实际上是用于存储线路图数据的方式。

集合的物理结构：集合的物理结构是使用qt中的Qset实现的，以便有效地维护站点与线路之间的关联关系。这个物理结构用于支持快速查找站点所属的线路以及某线路上有哪些站点的操作。

**2.5开发平台**

**开发平台：**

计算机型号 ：Lenovo Legion Y70002021

CPU ：11th Gen Intel® CoreTM i5-11400H @ 2.70GHz

操作系统 ：Windows 10

开发语言 ：C++(C++11标准以上)

开发框架 ：Qt Version 5.14.2

集成开发环境 ：Qt

编译器 ：Qt 5.14.2 (MinGW 7.3.0 32-bit)

**运行环境：**

可在上述集成环境下运行。

同时通过windeployqt.exe及Enigma Virtual Box进行整合压缩为一个sorting\_visulization\_boxed.exe文件，可在普通windows 机型（winsows 7以上版本）直接运行。

**开源代码的使用：**

本程序中的mouseviewzoom.h和mouseviewzoom.cpp参考了：

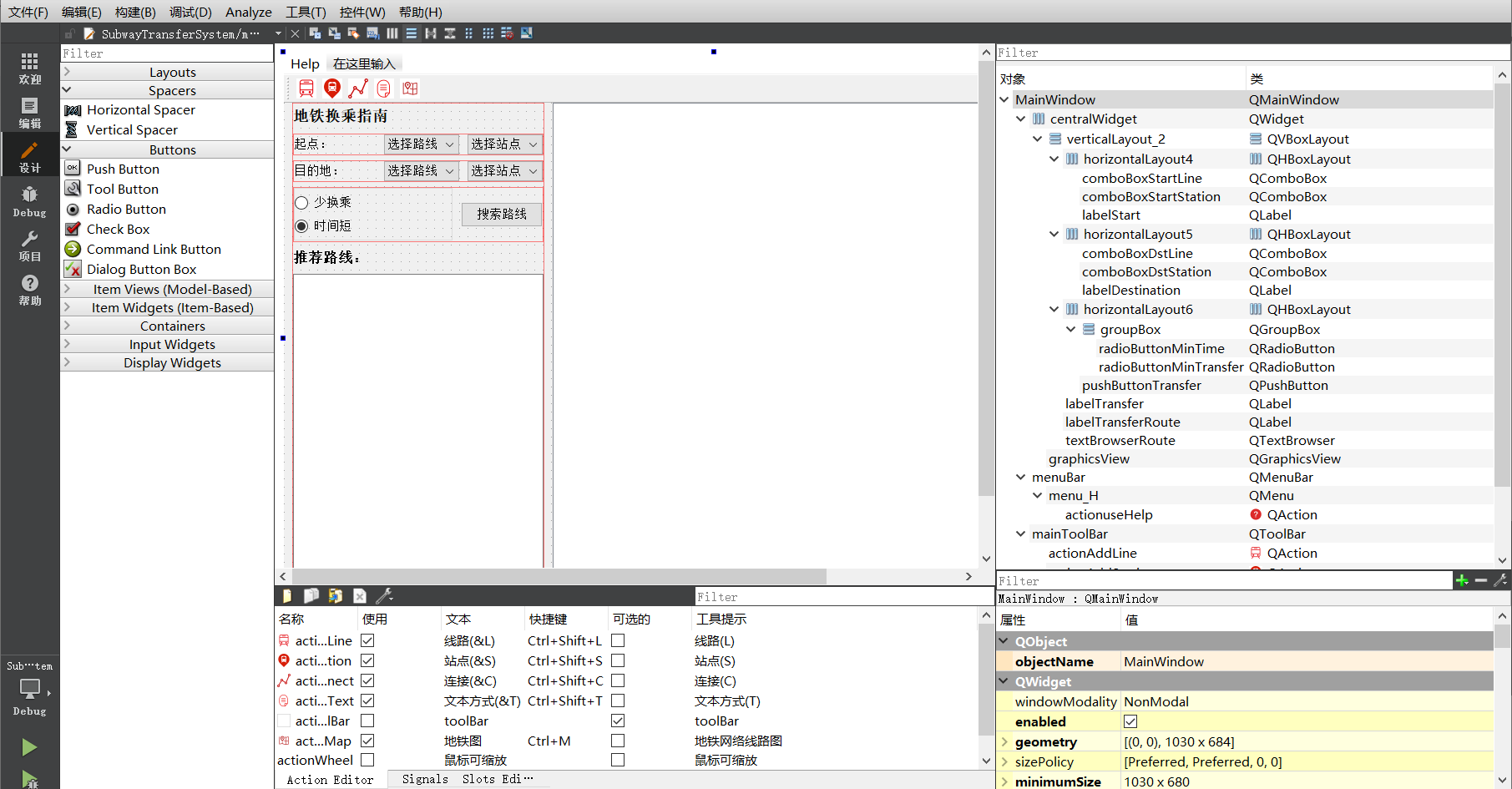
<https://stackoverflow.com/questions/19113532/qgraphicsview-zooming-in-and-out-under-mouse-position-using-mouse-wheel>

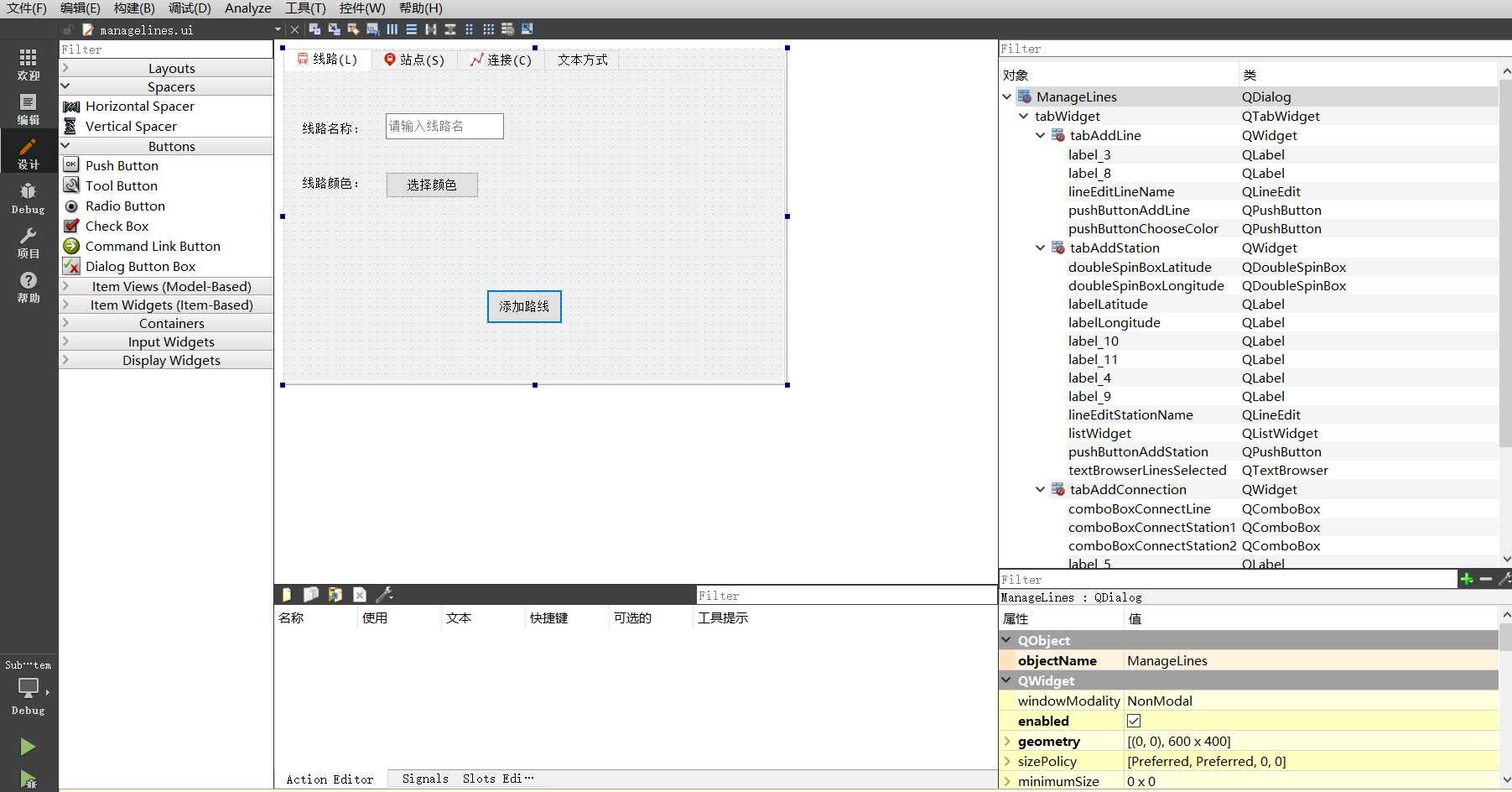
其功能是能够使用鼠标滚轮实现页面的缩放。

**2.6系统的运行结果分析说明**

**2.6.1调试及开发的过程**

**开发过程**中，先按照设计思路将各文件建好，之后先通过.ui文件绘制好界面：





之后再去定义各类中需要用到的私有变量以及成员函数，确定好类与类之间的继承与派生关系。

最后通过信号与槽函数机制，将前端与后端联系起来，实现前后端的同步。同时通过qt中的资源文件来向程序中插入icon图标以及加载html文件。

**调试过程**中，主要使用qt中的qdebug对中间变量进行输出以及配合qt中自带的调试功能来进行调试。

**2.6.2所开发软件达到的成果**

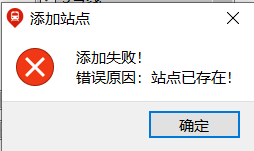
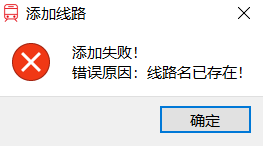
**正确性及稳定性**：

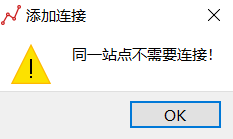
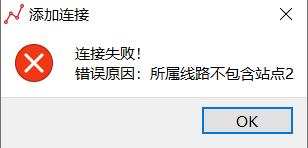
对于该程序，通过更换10多种不同的起点和终点，按照两种方式进行换乘查询，经过人工比对，程序都能给出正确的结果。同时程序能够稳定运行，在1s内给出换乘的结果，同时能够通过滚轮放大和缩小展示区的试图，比较方便；对于动态添加，分别尝试了添加一条新的线路，在已有的线路上添加新站点，在已有的两站建立新的线路，程序均能完成相应的动态添加。

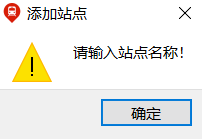
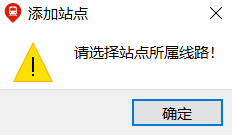
**容错性**：

程序具有较好的容错性，能够极大程度限制许多非法操作，下面通过几个操作进行说明。

①一些非法操作会进行提示：





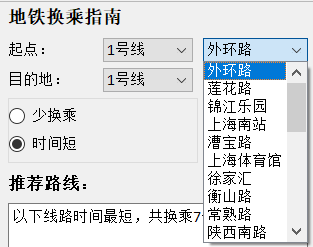


②利用控件保证输入正确性：



在添加站点时，输入的经纬度均设置好了范围限制。

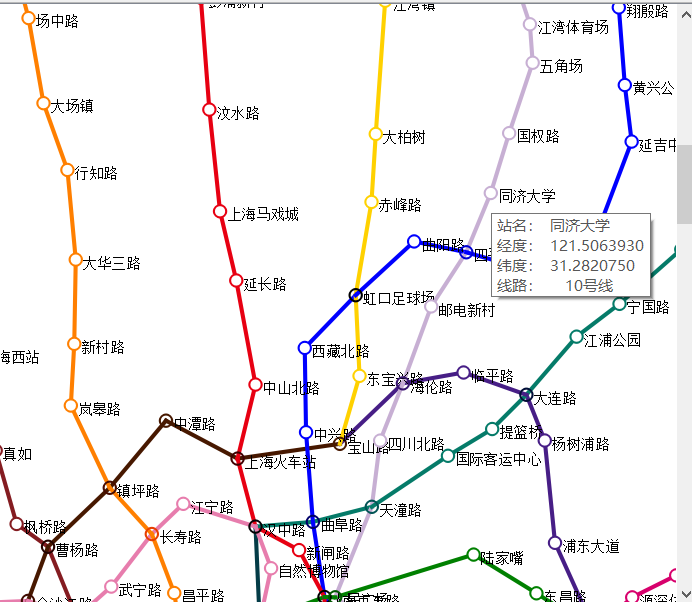
③用选择而非输入的方式避免了非法数据的出现



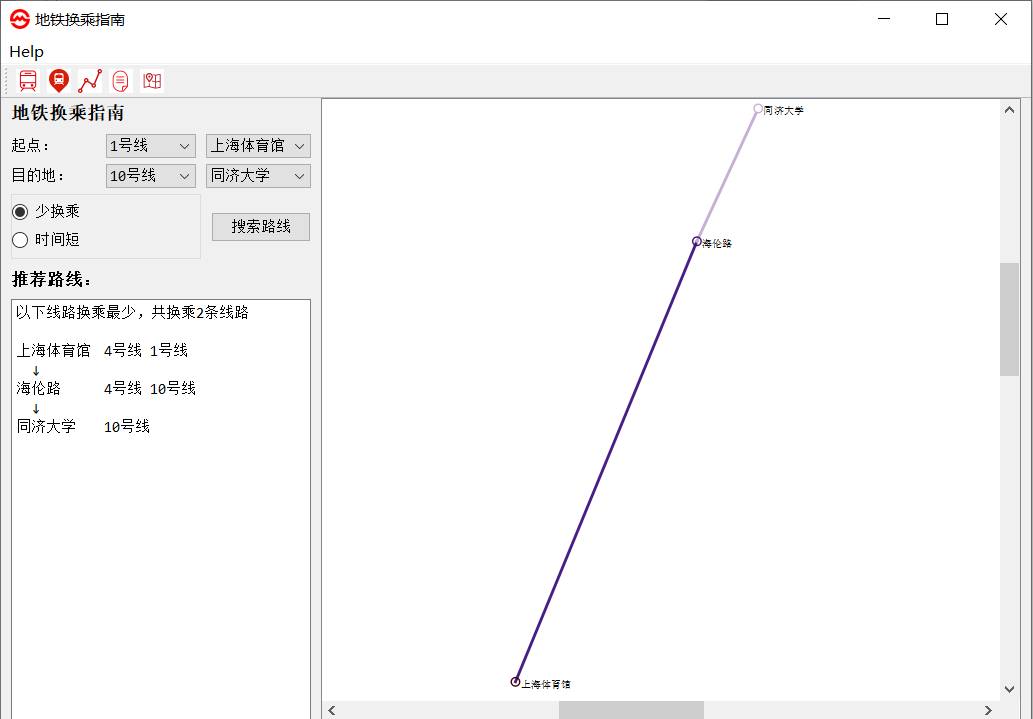
**2.6.3通过案例说明运行结果**

**换乘查询**：

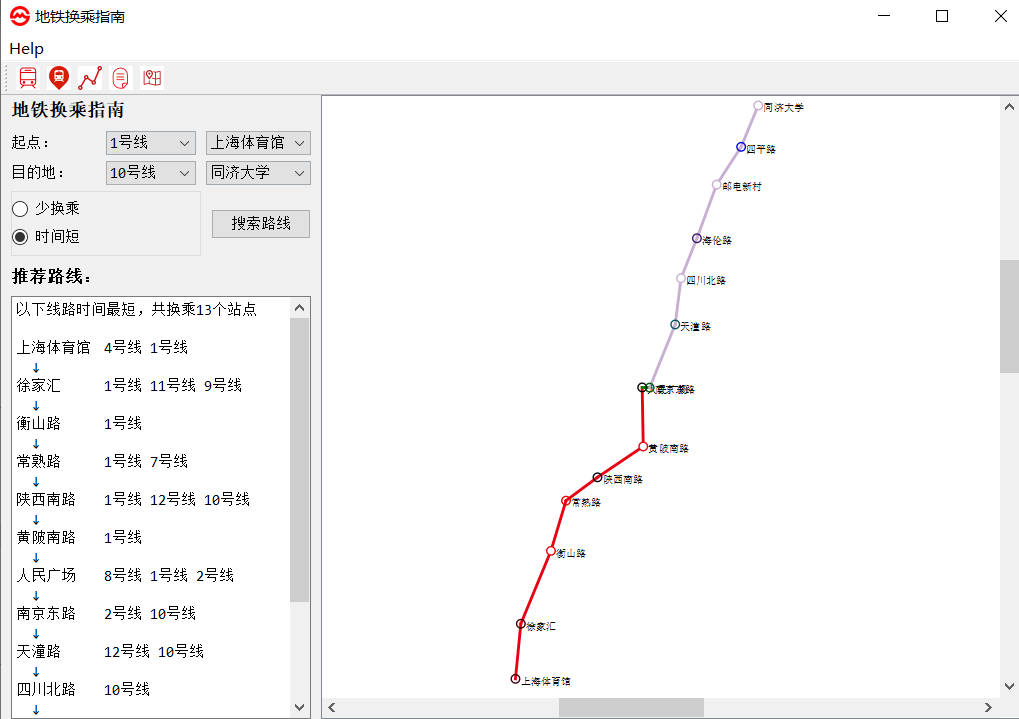
打开程序后地铁线路图呈现出来，可以将鼠标移动到站点上来查看该站点的信息。



以上海体育馆到同济大学为例，设置好起点和终点后，选择“少换乘”，查询结果如下：

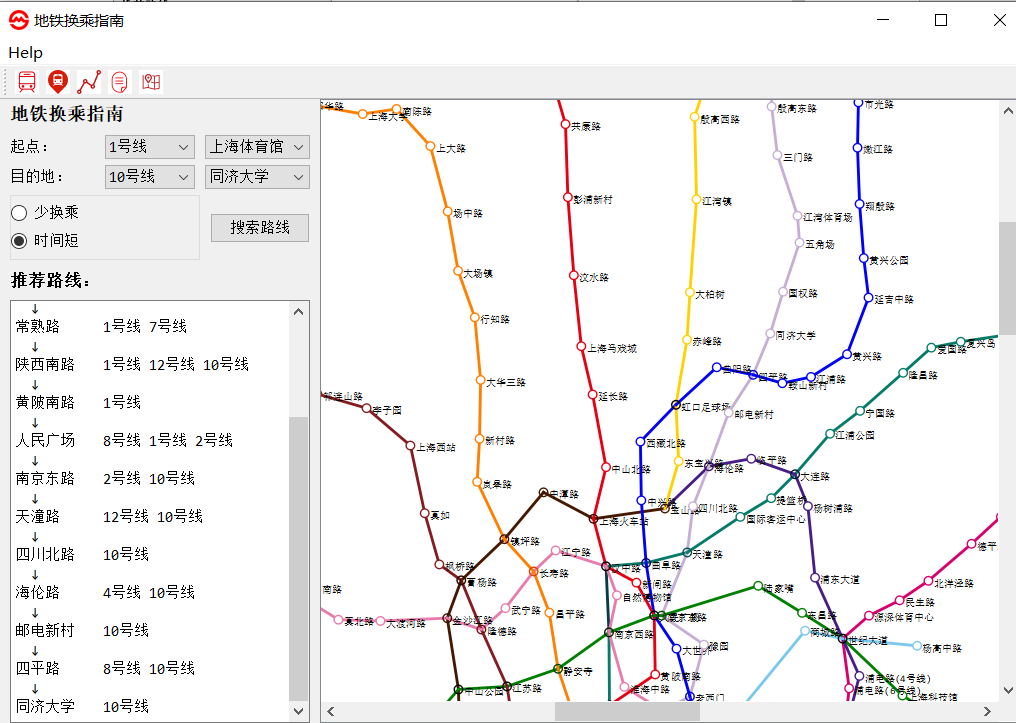


选择“时间短”，换乘查询如下：

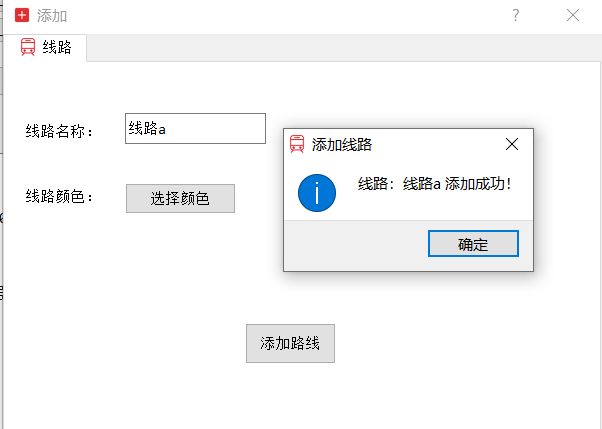


**动态添加**：

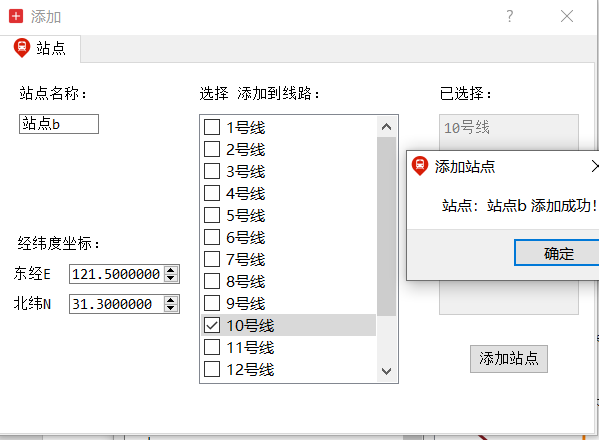
之后我们点击按钮，地铁网络图又再一次全部呈现。



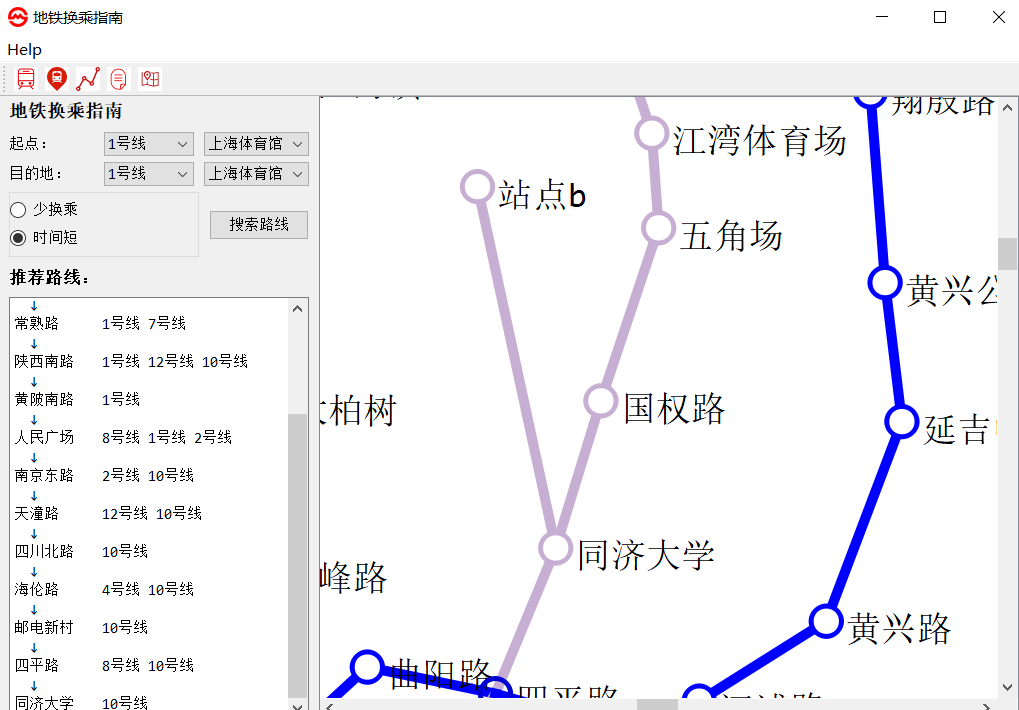
我们尝试添加线路：



添加站点：



再添加站点b与同济大学的连接，之后观察显示区：



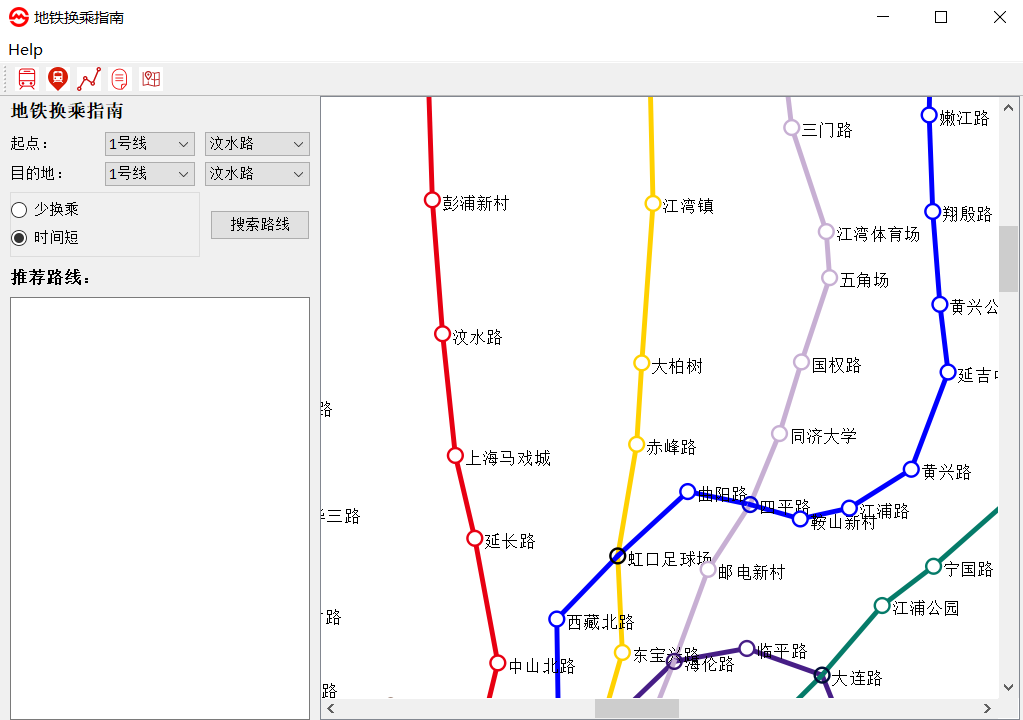
可见，“站点b”与“同济大学”在10号线上连接起来了

**2.7操作说明**

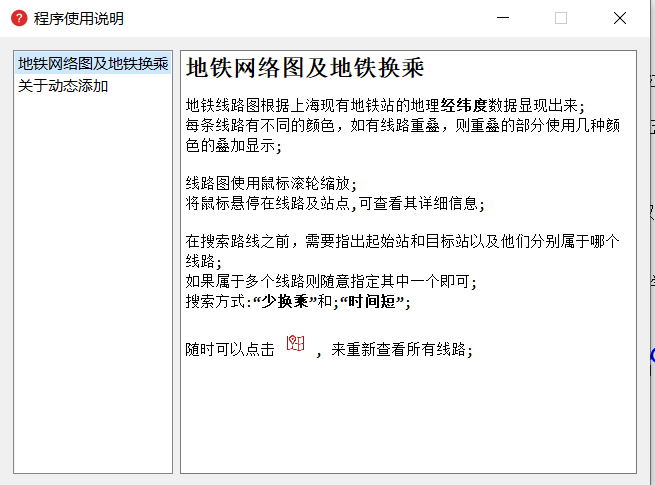
①首先打开程序后会显示如下的初始界面，持续2s左右



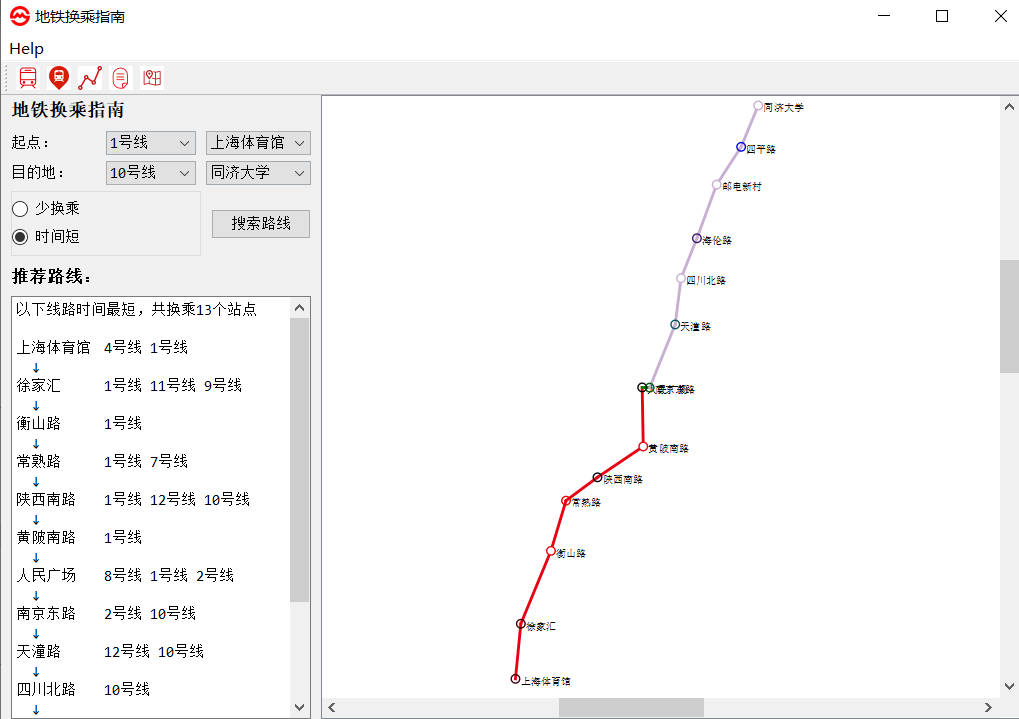
②之后进入如下的主界面：



③可以点击导航栏的Help来查看程序的使用说明：



④可以通过选择起点、目的地以及换乘方式来进行换乘查询：



⑤可以通过这5个按钮分别完成“动态添加线路”、“动态添加站点”、“添加连接”、“文本添加”、“重新显示所有线路图”，其中动态添加每个部分都有相应的提示和要求。



### 第三部分 实践总结

**3.1.所做的工作**

本次课程设计中，我完成了排序算法的可视化和地铁换乘系统项目。

首先，我学习了Qt5编程框架开发的相关知识，在前端界面上我学习了Qt中UI界面的设计，同时学习了qss样式文件的使用。在后端的存储上，我也进一步学习了如何将程序封装成类，让自己的程序更加清晰。

在排序算法可视化项目中，我设计了6种排序算法的可视化过程，程序界面简洁美观并具有友好性与较高的容错性，易于使用且提示较多，同时给出的各种排序信息和文字说明有利于更好的理解不同排序之间的特点和差异。

在地铁换乘项目中，我设计了一套地铁换乘系统，界面清晰，能够将上海市真实的地铁情况呈现出来，能够查询任意两站之间的时间最短和换乘最少的方案，而且能够动态的添加地铁线路及站台。在使用的过程中有很多提示以及限制，确保了程序的友好性和容错性。

**3.2.总结与收获**

**学习能力方面的提升**：

感觉自己的自学能力得到了提升，之前学习数据结构与算法的过程中，有老师的细心讲解。但本次完成项目中，Qt5开发框架都要靠自己自学，之后靠自己来实现整个项目。同时对于资源的搜索能力也提高了，程序中的icon文件都要从网上获得，同时程序中的部分代码也参考了部分网站。自学能力和资源搜索能力都是很重要的能力，在今后的学习中，我们必然会面临着大量的自学任务，同时我们处于一个开源的世界，合理利用别人的代码也相当重要，这两点能力的提升对我有很大的帮助，能够让我更加从容地面临之后学习中的挑战。

**课程设计的收获**：

虽然课程设计只有两个题目，但是从布置课程作业到完成两个项目的整个过程中，我学习到了以前从未接触过的标准用户交互程序，从一开始界面简陋、程序bug多，到最后界面美观、容错性强的程序，为我打开了编程的一扇新的大门。本次项目中，我不仅学习到了qt编程框架、ui模型绘制、文件操作等具体的知识；同时也让我真切地感受到实际应用中的项目有多么复杂，让我明白了在做项目时要求良好的编程习惯，学会借鉴他人的开源代码等等。当真正完成整个项目的时候真的很开心，通过自己解决一个又一个问题，让我树立起了不惧怕未知知识，勇于解决各种问题的信心。

**个人体会**：

本次项目从开始选择编程语言，到最后整个项目的完成，总共经历了三周半的时间，也有很多感触。

第一，在对于未知知识的学习上，我们应选择一些著名的文档与书籍，它们更加具有权威性，在学习qt的过程中，一开始也尝试了从b站找相关教学，但这些教学不够系统，不够专业。最后还是选择了官方的文档进行学习，并且通过实践的方式入门了qt。

第二，在项目的设计上，正所谓“磨刀不误砍柴工”，我们在完成一个项目的时候不应该上来就着急写代码，更应该先从宏观的角度进行构思，先将程序的思维导图画出来，自顶向下设计，分模块实现并且自底向上实现，这样能使我们更加有条理的完成项目。

第三，实际生活中的项目是十分困难的，本次地铁换乘系统已经不算简单了，但是还是有很多地方没有考虑进去。比如：每一辆地铁实时的位置、发车时间和速度，以及某些日子里有些地铁不会开通，不同地铁的早班车和末班车时间，某些地铁人可能已经满了。这些现实因素其实都是应该考虑进去才能设计出一个真正能在生活中使用的程序，本次的项目只能算是一个理想化的模型。

### 第四部分 参考文献

1. 霍亚飞.Qt Creator快速入门第二版[M].北京：北京航空航天大学出版社,2014-1
2. 霍亚飞，程梁. QT5 编程入门[M]. 北京：北京航空航天大学出版社,2015-1
3. 严蔚敏，吴伟民.数据结构（C语言版）[M].北京：清华大学出版社,2007
4. 大塚弘记.GitHub入门与实践[M].北京：人民邮电出版社,2015-7
5. Knuth, D. E. (1998). "The Art of Computer Programming, Volume 3: Sorting and Searching." Addison-Wesley Professional.
6. Sedgewick, R., & Wayne, K. (2011). "Algorithms." Addison-Wesley Professional.
7. 中国色 (Zhongguose). (2023). 竹绿、品蓝、茶花红.https://zhongguose.com
8. 阿里巴巴. Ant Design. <https://ant.design/components/overview-cn>
9. Qt Company. (2021). QWidget Class Documentation. In Qt Documentation. <https://doc.qt.io/qt-5/qwidget.html>
10. Strakhov, P. (2013). "QGraphicsView: Zooming in and out under mouse position using mouse wheel." Stack Overflow. <https://stackoverflow.com/questions/19113532/qgraphicsview-zooming-in-and-out-under-mouse-position-using-mouse-wheel>