Final_ Assignment: Melody of Sorting && A CLI Nefelibata

Jiaming Zhang[1] **and Xiaoling Zhang**[*]

***For correspondence:**
Jmzhang@bjfu.edu.cn

† 张家铭，学号：3210251。本次作业作者，包括成文、代码、创意在内，全部任务由本人的学习与实践完成。
* 作者未婚妻，在完成项目期间为作者提供了很多灵感与精神上的鼓励。（并非本校学生，未参与实际项目实现）
‡ 詩經．大雅．桑柔：「靡有旅力，以念穹蒼。」

Present address: † 林业装备与信息化, 工学院, 北京林业大学

[2] 数据可视化，结课作品集：排序算法可视化解释器 && 踱步命令行穹苍

Abstract

本次结课作业分为两个部分，第一部分为一个 Web 端的排序算法可视化解释器实现 (Melody of Sorting, MOS)，它包括了：冒泡排序、堆排序、快速排序等经典的排序算法，对于算法初学者理解算法工作流有一定辅助作用；为了使得排序效果更加清晰，加入了听觉元素，当排序成功后，会得到一段渐进高升无杂音的旋律。

而另一部分则是多个基于命令行的工具的集合 (CLI Nefelibata, CLIN)，他们共同形成了我的日常工作流。命令行接口对于服务器端工程来说不可或缺。而后端常用的操作系统如：Linux、UNIX 等往往只提供非常简陋的命令行交互界面，在一定程度上降低了用户体验感。本项目结合数据可视化课程思想，将单调的命令行序列解构，从实际应用价值角度以及个人使用经验两方面，对其进行重排设计。同时为了保持工作流（Workflow）一致性，自行开发了部分命令行工具，如基于 ASCII 码绘制的流程图绘制工具、类似 Obsidian 的交叉引用笔记、...、...、等。最终目的是将设计感与实用性融合，不仅愉悦了使用者自身的工作体验，一定程度上也提升了效率。

Contents

概述（Introduction）

「一」Melody of Sorting (MOS)

所谓排序，就是使一串记录，按照其中的某个或某些关键字的大小，递增或递减的排列起来的操作。排序算法，就是如何使得记录按照要求排列的方法。排序算法在很多领域得到相当地重视，尤其是在大量数据的处理方面。一个优秀的算法可以节省大量的资源。在各个领域中考虑到数据的各种限制和规范，要得到一个符合实际的优秀算法，得经过大量的推理和分析。

由于排序算法常常作为算法入门的第一课，另外排序算法大多没有用到很高深的理论知识，所以很多人对排序算法的学习并不够重视，并不了解其中的运作原理，导致了在实际写程序时，在一些细节上会犯错。其次，排序算法作为计算机初学者入门时接触的知识，部分算法思想脱离了高中时期的思考方式，导致部分学生会觉得抽象的算法难以学习。

综上，我觉得研发一款通过动画形式展现算法运行过程的小应用是十分必要的。鉴于 Web 端的优秀可移植性、可访问性以及轻量性，我最终放弃了使用纯命令行和基于 Qt 的方案，而是选择基于 Javascript 的形式来开发该应用。由于从未接触过前端三剑客（HTML，CSS，Javascript），所以借着这个项目可以驱动我带着一个目标快速的学习前端知识，扩充自己的业务栈。

理念设计

本项目的初衷是为了能够生动的展示排序算法的运行过程，并且使得该程序能够突破平台的限制，因此选择 Web 平台具有必然性。而为了体现生动性，自然应该选择动画的形式，具体到细节，则需要提炼出几大经典算法的共性、个人在学习中所遇到的难点等；概括来讲，我的初衷是写出一个：“我希望在我初学时，我所希望看到的一个教学工具”。

在具体实现的时候，我选择了一种投机的方式，将具体的交互流程抛给用户，而不事先设计一个针对不同算法的具体数据。这样做的好处是，用户具有较高的自由度，可以根据需要调整程序，自行探索不同算法的差异性。另一方面，程序的设计成本下降，我可以用更多的时间花在后端算法（关于本程序本身的算法）设计上。同时也防止我将本人的意志施加于使用者之上。

在规划此项目的时候，我注意到网络上已经有很多可视化排序算法的方案，其中不乏有很多很有创造性的想法。如1的方案，几乎和我的构想完全一致。

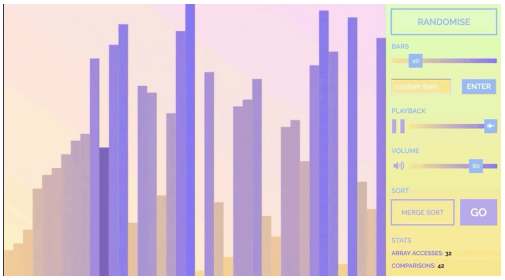

Figure 1. Avncharlie/ToneSort 的排序之声

因此不可避免的，我在实现的过程中借鉴了多处它的设计。其中包括布局、交互选项、以及类似彩虹的视觉设计等。但是代码的主体部分，并未照抄其代码。同时，Avncharlie/ToneSort 的方案中仅仅支持快速排序，而我的程序中，可供用户选择的排序算法包括：冒泡排序、插入排序、选择排序、鸡尾酒排序、快速排序、堆排序等九种排序算法。

另外，根据待排数组的尺寸不同，不一定只有彩虹形式的矩形更有视觉直观性，因此对于不同尺寸的数组，我设计了三种不同的显示方案如2所示：球形、单色矩阵、彩虹矩阵等。在此基础上，我设计了扫描指针的颜色变化、交换两个“元素”时的强化颜色变化效果。能够更加直观的体现出算法的运行过程，给用户带来更好的学习体验。

交互方式

MOS 程序使用起来非常直观。它包含许多排序算法，可以通过左侧的列表框进行选择。对于快速排序，可以单独选择枢轴选取规则。同时，也可以根据需要选择不同的待排数组初始状态、尺寸，来观察不同算法与算法在不同数据下的表现。

按“排序!”按钮时，对应的排序算法启动。再次点击将停止运行的算法，并且启动新的算法；当位于屏幕右侧的矩形按升序排列好时，程序停止运行。可以随时使用延迟控制、音量键来调整运行程序的表现形式。该程序将在默认音频输出上产生声音效果，当选取到相应的矩形时，程序将根据其代表的值

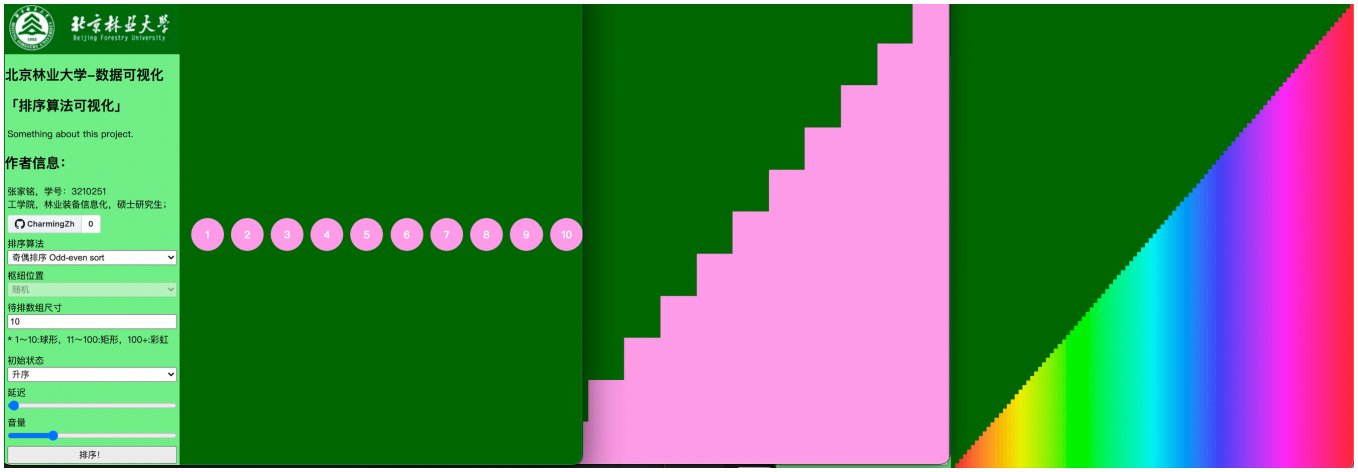

Figure 2. 球形、单色矩阵、彩虹矩阵三种显示效果

的大小，来输出较高或较低的音调（8-bits 风格电音）。当数组排序完成时，将会产生一段没有杂音的，音调由低至高的上升旋律。

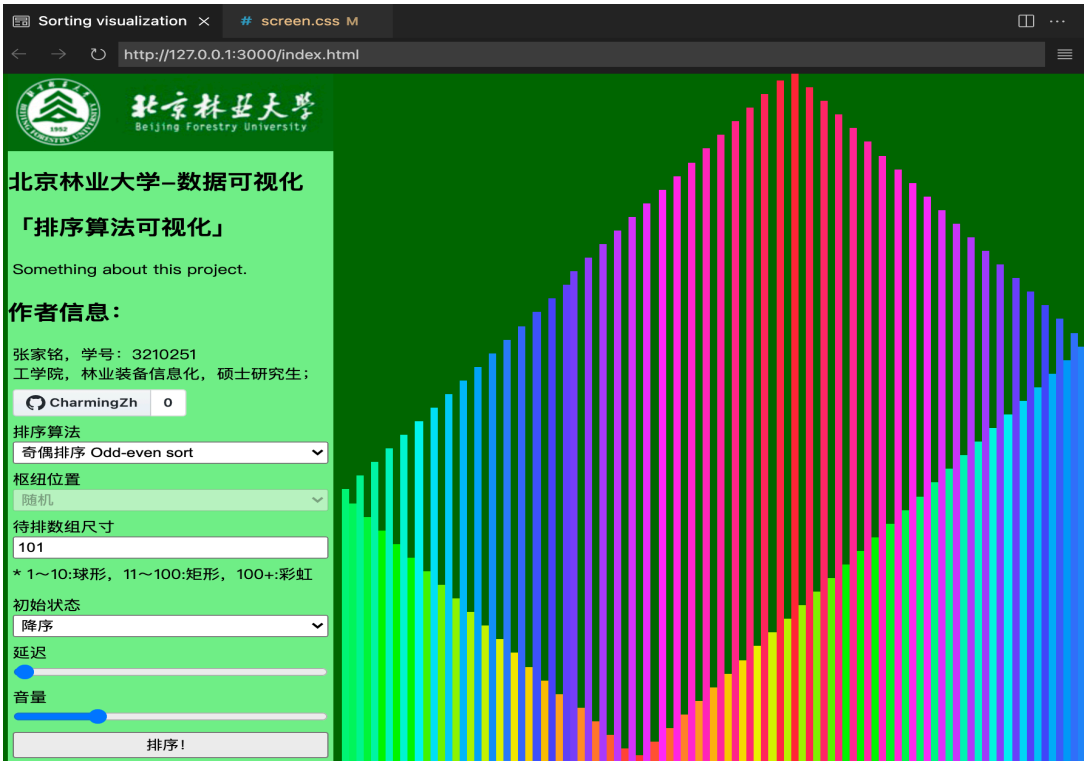

Figure 3. NOS 程序的主界面

「二」A CLI Nefelibata

基于 ASCII 码的流程图绘制工具、重新设计的 CLI 常用命令如：ls、cd、pwd、cat、tree 等，例如如下实例：

```
# 一个简单的注释程序，配合改版后的 cat 程序使用。
# 他可以生动的给源代码文件标记注释，同时不改变原程序
# 其原理为为使用程序时自动为源代码文件生成一个配套的".*"文件

    vim -p .bashrc .vimrc
```

```
    \_/ |  \_____________/
      |  |           |
      |  |           \- 打开哪个文件？
      |  |
      |  \- 使用tab模式
      |
      \- 打开源代码编辑器
```

整体框架

如图3所示，整个界面分为左侧控制区、右侧显示区。页面框架是静态的，使用 HTML 语句分割成两个区域，区域大小不需要做特别的处理，会根据行文字长度自动进行调整。在body区域尾部引入了JavaScript脚本，页面的动态效果是通过 JS 书写的；原理为通过数组排序为桥梁，每一次排序时都渲染出一个矩形或者球形，根据对应数组元素的值，自动生成长度；然后动态渲染出一个形状。数组元素的数值是唯一的，因此没有办法展现出排序算法的稳定性，但是方便渲染动画。

```
<body>
    <main id="content" class="content">
        <section id="configuration" class="">
            ...
        <\section>
        <section id="configuration" class="">
            ...
        <\section>
        ...
    </main>
<body>
```

动画中渲染的矩形长度与数组对应元素值是无关的，只作为表现相对大小用途。声称矩形的原理为：在读取到用户设置的待排数组大小后，声称该数量大小的数组，然后将其均匀的分割，每一个矩形的增量都是相通的。最后，再根据用户设置的数组初始状态，设置每一个元素的位置。由于数组值是唯一的，并且是等长递增的，因此可以通过生成与数组长度相同的随机数向量，作为数组各个元素的位置，来打乱数组。

```
        // 将 document 中的元素赋值给 JS 变量
        var algorithm = document.querySelector('#algorithm');
        var pivot = document.querySelector('#pivot');
        var shuffle = document.querySelector('#shuffle');
        var size = document.querySelector('#size');

        // 从用户输入中获取 待排数组尺寸
        var length = Number(size.value);
        // 根据尺寸新建一个数组，并且赋值
        var array = new Array(length);
        for (var i = 0; i < array.length; i++) {
          array[i] = i + 1;
        }

        // 设置待排元素 初始位置
        array.sort(function(a, b) {
          switch (shuffle.value) {
            case 'random':
              return Math.random() > 0.5 ? -1 : 1;
            case 'ascending':
              return a - b;
            case 'descending':
              return b - a;
          }
        });
```

在渲染长方形的时候，使用了一个for循环，在遍历生成渲染的同时，还可以设定位置。

```
var visualization = document.querySelector('#visualization');
        visualization.innerHTML = '';

        // 初始化长方形
        for (var i = 0; i < array.length; i++) {
          var element = document.createElement('span');

          var value = array[i];
          element.dataset.value = value;

          // 矩形高度
          var percent = (value / array.length) * 100;

          // 当待排数组尺寸不同时，展现出不同的效果
          if (array.length <= 10) {
            element.className = 'ball';
            element.innerText = array[i];
          } else {
            element.className = 'bar';
            element.style.height = percent + '%';
          }
          if (array.length >= 100) {
            // 当尺寸大于等于 100 时，使用彩虹效果
            element.style.backgroundColor = 'hsl(' + ((percent / 100) * 360) + ',
                                                    85%, 60%)';
          }

          visualization.appendChild(element);
        }
```

颜色选取了北京林业大学的标志性绿色，以此为主色调为其他附属元素选取相近的配色（有点丑，我知道）。矩形使用了炫酷的粉色。这方面的工作主要使用前端工具，在 CSS 文件中自动生成。值得注意的是，当待排数组的大小不同时，会采取不同的渲染样式。

具体工作

排序算法实现

由于对前端工作不是很熟悉，所以JavaScript编程的学习是一个重点工作。而由于编程思想大多是相近的，而我的实现也是基于一个数组的排序，在渲染效果。因此，学习了基础语法后很容易实现出其他语言实现过的轮子。由于篇幅限制，我只展现出了一个最基本的冒泡排序实现，以及快速排序的JavaScript实现，其余的七种排序方法都放在了源文件中供老师检查。

冒泡排序

```
function bubbleSort(a) {
  var n = a.length;
  var sorted;
  for (var i = 0; i < n; i++) {
    sorted = true;
    for (var j = 0; j < n - i - 1; j++) {
      if (test(a, j + 1, j) < 0) {
        sorted = false;
        swap(a, j, j + 1);
      }
    }
    if(sorted == true){
        break;
    }
  }
}
```

快速排序

```
function pivot(aa, type, left, right) {
  if (typeof(left) === 'undefined') left = 0;
  if (typeof(right) === 'undefined') right = aa.length() - 1;
  var p = null;
  if (type === 'random') {
    var p = left + Math.floor((right - left + 1) * Math.random());
  } else if (type === 'first') {
    p = left;
  } else if (type === 'last') {
    p = right;
  } else if (type === 'middle') {
    p = Math.round((left + right) / 2);
  } else {
    throw new TypeError('Invalid p type ' + type);
  }

  return p;
}

function partition(aa, type, left, right) {
  var p = pivot(aa, type, left, right);
  swap(aa, p, right);

  p = left;
  for (var i = left; i < right; i++) {
    if (test(aa, i, right) < 0) {
      if (i != p) {
        swap(aa, i, p);
      }
      p += 1
    }
  }

  swap(aa, right, p);

  return p;
}

function quickSort(aa, type, left, right) {
  var n = aa.length;
  if (typeof(left) === 'undefined') left = 0;
  if (typeof(right) === 'undefined') right = n - 1;

  if (left >= right) return;

  var p = partition(aa, type, left, right);
  quickSort(aa, type, left, p - 1);
  quickSort(aa, type, p + 1, right);
}
```

requestAnimationFrame 模块的引入

用 js 实现一个无限循环的动画。首先想到的是定时器。至于时间间隔为什么是 1000/60, 这是因为大多数屏幕渲染的时间间隔是每秒 60 帧。既然 setInterval 可以搞定为啥还要用 requestAnimationFrame 呢？最直观的感觉就是，添加 api 的人是个大神级牛人，我只能怀疑自己。所以搜索相关问题发现以下两点：

- requestAnimationFrame 会把每一帧中的所有 DOM 操作集中起来，在一次重绘或回流中就完成，并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率，一般来说，这个频率为每秒 60 帧。

- 在隐藏或不可见的元素中，requestAnimationFrame 将不会进行重绘或回流，这当然就意味着更少的的 cpu，gpu 和内存使用量。

我没有添加各个浏览器的兼容写法，这里只说用法。效果是实现了，不过我想到两个问题。首先是，怎么停止 requestAnimationFrame？是否有类似 clearInterval 这样的类似方法？答案是确定的必须有：cancelAnimationFrame() 接收一个参数 requestAnimationFrame 默认返回一个 id，cancelAnimationFrame 只需要传入这个 id 就可以停止了。

另外就是，如果我想动画频率降低怎么做，为什么不考虑加快？当前刷新频率已经是屏幕的刷新频率了，再快也没有意义了。默认情况下，requestAnimationFrame 执行频率是 1000/60, 大概是 16ms 多执一次。如果我们想每 50ms 执行一次怎么办呢？requestAnimationFrame 执行条件类似递归调用，既然这样的话我们能否自定一个时间间隔再执行呢？当然定时器这么 low 的东西我们就不考虑了，都已经抛弃它用 rAF 了。解决方案是，当和服务端通信时，记录下一个时间差，（时间差等于服务端时间-本地时间）不管正负我们只要这个时间差。这样每当我们接受到消息或者发送消息的时候我们就拿本地时间和是价差相加。

总结

项目自我评价

本项目在实现上难度不大，在建立起对于前端知识的简单了解后，剩余的主要是设计部分。尤其是在已经有人造过轮子的情况下，重复实现的目的主要是自我的学习。

核心部分与难点

- 前端页面的构建
- 动态低延迟渲染动画
- 可视化的形式设计
- 交互性的用户体验
- 可复用接口的设计，降低了开发的成本

体会心得

在完成此项目的过程中，我深刻的体会到了视觉可视化课程的意义，不同的感官刺激形式会带来客户不同的反馈体验，对于中国学生来说，大多数人学习计算机的途径为通过经典教材 + 刷题来完成，但是这种形式欠缺的了直观形象的解释性。我最初的目标是做出一个我在初学计算机时希望得到的一个工具。

虽然数据可视化课程直观上的扣题应该是做一个类似数据分析的表格、饼图云云，或是小组合作copy一个网上现成的项目，来混分数。而我不想把自己的生命浪费在无谓的事情上，我希望我做的每一件事都能对我的生活带来积极的改变，或是让我学到一些什么。从广义上来讲，数据无非是承载着信息的载体，可视化便是将其以直观的形式展现出来。数据结构可视化当然是一种数据可视化的形式，因此我的项目在我看来是一个很有创意的项目。

对于一个喜欢技术的人而言，尤其是冷门的技术，如果能够对日常工作流有积极的改变，则对于我来说更有吸引力。虽然这个项目并没有做的很完美，很多工作没有思路去完成，但是他显然已经成为了一个有用的工具，而不是从本次作业之后就在文件夹里落灰。所以，我对于我的作业十分满意。

Acknowledgments

感谢张晓灵同学牺牲掉假期玩耍时间陪我一起在学校自习，完成了这组项目。感谢王老师在课堂上生动、引人思考的讲解，启发我走过了很多思想上的局限。感谢我的家人，理解我在学业上的压力，不能及时回去。

References

Buchanan S, Ochs B, LaViola Jr JJ. CSTutor: a pen-based tutor for data structure visualization. In: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*; 2012. p. 565–570.

313 **Chen T**, Sobh T. A tool for data structure visualization and user-defined algorithm animation. In: *31st Annual*
314 *Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No.*
315 *01CH37193)*, vol. 1 IEEE; 2001. p. TID–2.

316 **Crescenzi P**, Malizia A, Verri MC, Díaz P, Aedo I. Integrating algorithm visualization video into a first-year algo-
317 rithm and data structure course. Journal of Educational Technology & Society. 2012; 15(2):115–124.

318 **Erkan AS**, VanSlyke T, Scaffidi TM. Data structure visualization with LaTeX and Prefuse. ACM SIGCSE Bulletin.
319 2007; 39(3):301–305.

320 **Halim S**. Visualgo–visualising data structures and algorithms through animation. Olympiads in Informatics.
321 2015; 9:243–245.

322 **Petit G**, Kornreich C, Noël X, Verbanck P, Campanella S. Alcohol-related context modulates performance of
323 social drinkers in a visual Go/No-Go task: a preliminary assessment of event-related potentials. PloS one.
324 2012; 7(5):e37466.

325 **Simson R**, Vaughan Jr HG, Ritter W. The scalp topography of potentials in auditory and visual Go/NoGo tasks.
326 Electroencephalography and clinical neurophysiology. 1977; 43(6):864–875.

327 **Yin H**. ViSOM-a novel method for multivariate data projection and structure visualization. IEEE Transactions
328 on Neural Networks. 2002; 13(1):237–243.