

**课 程 实 践 报 告**

|  |  |
| --- | --- |
| 学 院： | 珠海科技学院计算机学院 |
| 专业名称： | 软件工程 |
| 设计科目： | 数据结构课程实践 |
| 学号姓名： | 04222119张子恒 |
| 指导教师： | 郎六琪 |
| 完成时间： | 2024.6.1 |

[1.题目内容 2](#_Toc168567970)

[2.详细分析 2](#_Toc168567971)

[2.1 数据结构和算法说明 2](#_Toc168567972)

[2.1.1 冒泡排序（Bubble Sort） 2](#_Toc168567973)

[2.1.2 堆排序（Heap Sort） 3](#_Toc168567974)

[2.1.3 归并排序（Merge Sort） 4](#_Toc168567975)

[2.1.4 直接插入排序（Insertion Sort） 4](#_Toc168567976)

[2.1.5 选择排序（Selection Sort） 5](#_Toc168567977)

[2.1.6 快速排序（Quick Sort） 5](#_Toc168567978)

[2.2 设计思路 6](#_Toc168567979)

[2.3 类/接口/函数设计 9](#_Toc168567980)

[2.3.1 output.py 9](#_Toc168567981)

[2.3.2 data.py 10](#_Toc168567982)

[2.3.3 排序算法 11](#_Toc168567983)

[2.4 类/接口/函数关系 12](#_Toc168567984)

[3. 程序实现 12](#_Toc168567985)

[3.1核心代码说明 12](#_Toc168567986)

[3.1.1 output.py 12](#_Toc168567987)

[3.1.2 排序动画 16](#_Toc168567988)

[3.2程序运行 17](#_Toc168567989)

[4.测试与分析 20](#_Toc168567990)

[5. 结论与心得 21](#_Toc168567991)

[6.参考资料 22](#_Toc168567992)

1.题目内容

排序数据随机产生，针对随机案例，对冒泡排序、堆排序、归并算法，提供排序执行过程的动态图形演示。

2.详细分析

2.1 数据结构和算法说明

### 2.1.1 冒泡排序（Bubble Sort）

冒泡排序是一种简单直观的排序算法。它重复地遍历要排序的列表，比较相邻的两个元素，并且交换它们的位置，直到整个列表排序完成。具体步骤如下：

1. 从列表的第一个元素开始，依次比较相邻的两个元素。
2. 如果第一个元素比第二个元素大（或者小，取决于升序还是降序），则交换它们的位置。
3. 继续向后遍历列表，重复上述比较和交换步骤，直到列表末尾。
4. 重复以上步骤，每次遍历都会将当前未排序部分的最大（或最小）元素“冒泡”到列表的末尾。
5. 重复以上步骤，直到整个列表排序完成。

冒泡排序的时间复杂度为 O(n^2)，空间复杂度为 O(1)。尽管冒泡排序实现简单，但在实际应用中往往效率较低，特别是对于大型数据集。

### 2.1.2 堆排序（Heap Sort）

堆排序是一种利用堆数据结构的排序算法。它利用了堆的特性，将待排序的序列构造成一个堆（通常是大顶堆），然后依次将堆顶元素与末尾元素交换，重新调整堆，直到整个序列有序。具体步骤如下：

1. 构建初始堆：将待排序的序列构造成一个大顶堆。
2. 交换堆顶元素和末尾元素：将堆顶元素（当前最大元素）与末尾元素交换，并将堆的大小减 1。
3. 重新调整堆：将剩余元素重新调整成大顶堆。1
4. 重复步骤 2 和步骤 3，直到堆的大小为 1。

堆排序的时间复杂度为 O(n log n)，空间复杂度为 O(1)。它具有原地排序的特点，不需要额外的空间，因此在空间效率上比较高。

### 2.1.3 归并排序（Merge Sort）

归并排序是一种分治算法，它将待排序数组分成两个子数组，然后递归地对子数组进行排序，最后将排好序的子数组合并成一个整体有序的数组。具体步骤如下：

1. 分解：将待排序的数组递归地分解成两个子数组，直到每个子数组只有一个元素。
2. 合并：将两个有序的子数组合并成一个有序数组，同时进行排序。
3. 重复以上步骤，直到所有子数组合并完成，得到最终的有序数组。

归并排序的时间复杂度为 O(n log n)，空间复杂度为 O(n)，因为需要额外的空间来存储临时数组。尽管归并排序在空间上略高，但由于其稳定性和适用性，仍然被广泛应用。

### 2.1.4 直接插入排序（Insertion Sort）

直接插入排序是一种简单直观的排序算法，它通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。具体步骤如下：

1. 从第一个元素开始，该元素可以认为已经被排序。
2. 取出下一个元素，在已经排序的元素序列中从后向前扫描。
3. 如果已排序的元素大于新元素，将该已排序元素移到下一位置。
4. 重复步骤3，直到找到已排序的元素小于或等于新元素的位置。
5. 将新元素插入到该位置中。
6. 重复步骤2-5。

时间复杂度：最坏情况 O(n^2)，最好情况 O(n) 空间复杂度：O(1) 特点：适合于少量数据的排序，稳定。

### 2.1.5 选择排序（Selection Sort）

选择排序是一种简单直观的排序算法，它的工作原理是每次从待排序的数据中选出最小（或最大）的一个元素，存放在序列的起始位置，直到全部待排序的数据元素排完。具体步骤如下：

1. 初始状态：无序区为 R[1..n]，有序区为空。
2. 第i趟排序 (i=1,2,3…n-1) 开始时，当前有序区和无序区分别为 R[1..i-1] 和 R(i..n)。该趟排序从当前无序区中选出关键字最小的记录 R[k]，将它与无序区的第一个记录 R交换，使 R[1..i] 和 R[i+1..n] 分别变为记录个数增加1个的新有序区和记录个数减少1个的新无序区。
3. n-1趟结束，数组有序化。

时间复杂度：O(n^2) 空间复杂度：O(1) 特点：简单直观，适合数据量小的情况，不稳定。

### 2.1.6 快速排序（Quick Sort）

快速排序是一种分治算法，通过一次排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序。具体步骤如下：

1. 从数列中挑出一个元素，称为“基准”（pivot）。
2. 重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。
3. 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

时间复杂度：最坏情况 O(n^2)，平均情况 O(n log n) 空间复杂度：O(log n)（递归调用造成的栈空间） 特点：速度快，效率高，适合大规模数据的排序，不稳定。

2.2 设计思路

1. **目标设定**：

创建一个可视化工具，展示不同排序算法的执行过程,让用户能够直观地理解排序算法的工作原理。

1. **技术选型：**
   * 选择Python语言，因其简洁易懂，拥有丰富的库支持。
   * 选择Matplotlib库，不仅支持数据可视化，还能生成动画，与FFMpeg结合后能直接输出视频格式。
2. **排序算法实现：**
   * 采用Python实现多种排序算法，如冒泡排序、归并排序、堆排序等。
   * 每种排序算法都设计为能够记录其执行过程中的“帧”。
3. **动画帧处理：**
   * 在排序算法的关键步骤截取数据状态，作为动画的一帧。
   * 使用深拷贝来保存每一帧的状态，确保动画的流畅性和数据的一致性。
4. **颜色编码：**
   * 为了突出显示排序过程中的重要操作，采用颜色编码来增强视觉效果。例如，在交换元素时，将涉及的元素标记为特定颜色。
5. **动画展示：**
   * 利用Matplotlib的动画功能，将排序过程的帧连续播放，形成动画。
   * 提供播放、保存为HTML或MP4文件的功能，以适应不同的展示需求。
6. **用户交互：**
   * 允许用户通过命令行参数选择排序算法和数据类型。
   * 支持用户自定义动画的帧间隔和每秒帧数（fps）。
   * 用户可以通过命令行界面选择播放动画、保存为视频或HTML文件
7. **代码实现：**
   * 对于每种排序算法，实现一个函数，该函数不仅执行排序，还生成动画帧。
   * 使用注释明确区分动画帧操作部分，保持代码的清晰和可维护性。
8. **扩展性：**
   * 设计时考虑到扩展性，方便未来添加更多的排序算法和展示特性。

项目程序运行流程如图2-1

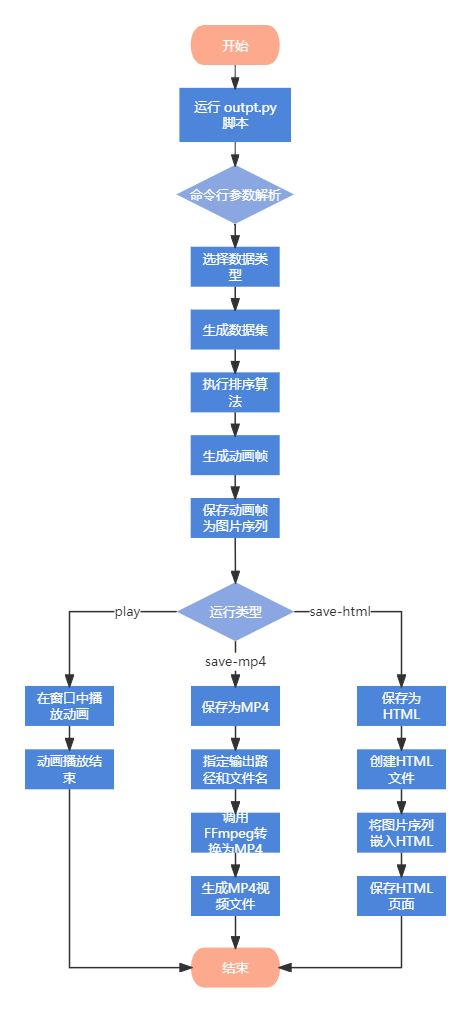


图2-1运行流程图

1. **开始：**用户启动排序算法可视化过程。
2. **运行 output.py 脚本：**用户在命令行中运行 output.py 脚本，开始执行排序算法的可视化。
3. **命令行参数解析：**脚本首先解析用户提供的命令行参数，这些参数决定了可视化的类型（播放、保存为 HTML 或 MP4）和排序算法的具体选项。
4. **选择数据类型：**根据用户的选择，脚本确定要生成的数据集类型，如随机、几乎排序好、少数唯一值或逆序。
5. **生成数据集：**脚本使用选定的数据类型生成一个待排序的数据集。
6. **执行排序算法：**脚本对生成的数据集执行用户选择的排序算法。
7. **生成动画帧：**在排序过程中，脚本记录每一步的状态，生成动画帧。
8. **保存动画帧为图片序列：**脚本将生成的动画帧保存为一系列图片文件。
9. **播放动画（如果用户选择 play）：**
   * 脚本使用 Matplotlib 在窗口中播放排序过程的动画。
10. **保存为 HTML（如果用户选择 save-html）：**
    * 脚本创建一个 HTML 文件。
    * 将图片序列嵌入到 HTML 文件中。
    * 保存 HTML 页面，用户可以在浏览器中查看动画。
11. **保存为 MP4（如果用户选择 save-mp4）：**
    * 脚本指定输出路径和文件名。
    * 调用 FFmpeg 命令行工具，将图片序列转换为 MP4 视频文件。
    * 生成 MP4 视频文件。
12. **结束：**完成动画的播放或保存后，程序结束

2.3 类/接口/函数设计

### 2.3.1 output.py

这个脚本是一个完整的应用程序，它展示了如何使用Python和matplotlib来创建教育性的排序算法可视化工具。用户可以通过命令行与程序交互，选择不同的排序算法和数据类型来观察排序过程。程序支持将排序过程保存为视频文件或HTML文件，方便分享和查看。

**全局变量：**

* **stype\_dic**：一个字典，存储排序算法名称和它们对应的索引。
* **titles**：一个列表，存储排序算法的标题和时间复杂度。
* **funs**：一个列表，存储排序算法的函数，可以通过索引快速访问。

**函数定义：**

* **create\_original\_data(dtype)**：根据提供的类型（"random", "reversed", "few-unique", "almost-sorted"）生成原始数据集。
* **draw\_chart(stype, original\_data, frame\_interval)**：绘制指定排序算法的动态图表。
* **draw\_all\_charts(original\_data, frame\_interval)**：绘制所有排序算法的动态图表。

**主程序：**

* 脚本首先尝试从用户那里获取要排序的数据项数量，如果失败则默认为32。
* 根据命令行参数，脚本可以执行三种操作：播放动画（play）、保存为HTML（save-html）或保存为MP4文件（save-mp4）。
* 根据用户输入的排序算法类型和数据类型，脚本调用相应的函数生成动画并展示或保存。

### 2.3.2 data.py

用于在排序算法可视化中表示和区分数据元素。通过动态设置颜色，它可以增加可视化的可读性和吸引力。

**全局变量：**

**data\_count（int）**:表示数据集中元素的总数,固定为32。

**构造函数：**

**\_\_init\_\_(self, value)**:初始化数据元素的值，并调用 set\_color 方法来设置颜色。

**类方法：**

**set\_color(self, rgba=None)**: 设置数据元素的颜色。如果未提供 rgba，则根据元素的值自动计算颜色。

### 2.3.3 排序算法

* **冒泡排序bubblesort.py**

执行冒泡排序，并通过深拷贝（copy.deepcopy）在每个关键步骤保存数据状态，形成动画帧列表（frames）。在每次元素交换时，将当前状态添加到帧列表，并标记交换的元素为红色，以突出显示。

* **堆排序heapsort.py**

**heap\_sort(data\_set):** 执行堆排序算法，并记录动画帧,返回一个包含排序过程每一步骤的动画帧列表

**heap\_adjust(ds, head, tail, frames):** 对堆进行调整，确保堆的性质得到满足

* **归并排序mergesort.py**

**merge\_sort(data\_set):** 执行归并排序算法，并记录动画帧。返回一个包含排序过程每一步骤的动画帧

**split\_merge(ds, head, tail, frames):** 递归地分割数据集，并合并已排序的子序列。

* **直接插入排序insertionsort.py**

执行插入排序算法，记录动画帧。返回一个包含排序过程每一步骤的动画帧列表。

* **选择排序selectionsort.py**

执行选择排序算法，记录动画帧。返回一个包含排序过程每一步骤的动画帧列表。

* **快速排序quicksort.py**

**quick\_sort(data\_set):** 作为快速排序的入口函数，初始化帧列表并调用递归的快速排序函数 qsort。返回一个包含排序过程每一步骤的动画帧列表。

**qsort(ds, head, tail, frames):** 递归函数，执行快速排序的分区操作，并递归地对子部分进行排序。

2.4 类/接口/函数关系

项目接口、调用关系如图2-2

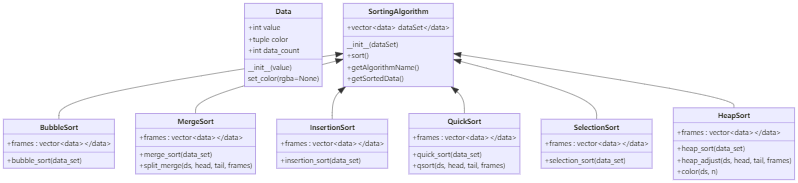


图2-2接口关系图

3. 程序实现

3.1核心代码说明

列出核心功能的相关代码，并加以说明。这里的核心代码可以是关键算法的代码，可以是通过自学实现的新技术代码，也可以是重要功能的相关代码等。注意，详细说明不仅指的是详细的注释，同时也要对相应代码涉及的原理和作用进行适当说明。不要粘贴全部源代码，请用截图的方式截取关键核心代码！文件排版要做的整洁干净舒适。

### 3.1.1 output.py

在创建原始数据集的函数 create\_original\_data 中，核心功能是根据提供的类型 dtype 生成一个特定顺序的整数列表，这个列表将用于后续的排序算法可视化。

**以下是核心代码的说明：**

|  |
| --- |
| def create\_original\_data(dtype):  data = []  if dtype == "random": # 对随机序列进行排序  data = list(range(1, Data.data\_count + 1)) # 生成序列 1 到 Data.data\_count 的整数列表  random.shuffle(data) # 随机打乱列表  elif dtype == "reversed": # 按降序排序  data = list(range(Data.data\_count, 0, -1)) # 生成降序列表  # ... 省略其他条件分支 ...  return data |

**核心功能的原理和作用：**

1. **随机序列** ("random"):

* 模拟大多数实际应用场景，元素初始状态无序。
* 使用 range(1, Data.data\_count + 1) 生成一个从 1 到 Data.data\_count 的整数序列。
* 调用 random.shuffle(data) 将列表中的元素随机打乱，为排序算法提供一个随机的初始状态。

1. **降序序列** ("reversed"):

* 测试排序算法处理完全逆序的数据集的能力。
* 使用 range(Data.data\_count, 0, -1) 生成一个从 Data.data\_count 递减到 1 的整数序列，形成一个降序排列。

1. **部分重复序列** ("few-unique"):

* 模拟具有大量重复元素的数据集，测试排序算法的效率。
* 将数据集划分为四部分，每部分包含数据集中的四分之一元素，每部分的元素值分别为 d, d \* 2, d \* 3，其中 d = Data.data\_count // 4。
* 然后随机打乱整个列表，创建一个包含部分重复元素的数据集。

1. **几乎已排序序列** ("almost-sorted"):

* 测试排序算法处理接近有序的数据集的性能。
* 首先生成一个从 1 到 Data.data\_count 的升序列表。
* 随机选择两个位置 a 和 b，如果相同则重新选择 b，保证 a != b。
* 交换这两个位置的元素，从而创建一个除了一对元素外都已排序的数据集。

在 **draw\_chart 函数**中，核心功能是生成一个动态图表来展示特定排序算法的过程。

以下是关键代码段的说明：

1. **初始化图表和子图**：

|  |
| --- |
| * fig = plt.figure(1, figsize=(16, 9)) axs = fig.add\_subplot(111) axs.set\_xticks([]) axs.set\_yticks([]) plt.subplots\_adjust(left=0.01, bottom=0.02, right=0.99, top=0.95, wspace=0.05, hspace=0.15) |

* 创建一个新图表，并设置为16:9的尺寸。
* 添加一个子图。
* 隐藏坐标轴刻度，以便专注于排序动画。

1. **转换原始数据并初始化动画帧**：

|  |
| --- |
| * data\_set = [Data(d) for d in original\_data] frames = funs[stype](data\_set) |

* 将原始数据转换为 Data 对象，这些对象包含了排序所需的所有信息，如值和颜色。
* 调用排序算法函数，获取整个排序过程的动画帧。

1. **打印排序算法信息**：

|  |
| --- |
| * print("%s: %d frames." % (re.findall(r"\w+ Sort", titles[stype])[0], len(frames))) |

* 打印排序算法的名称和动画帧总数。

1. **定义动画函数**：

|  |
| --- |
| * def animate(fi):  # ... 动画绘制代码 ... |

* 根据当前帧编号 fi 绘制柱状图，表示排序算法的当前状态。

1. **创建动画对象**：

|  |
| --- |
| * anim = animation.FuncAnimation(fig, animate, frames=len(frames), interval=frame\_interval) |

* 使用 FuncAnimation 创建动画对象，指定动画函数、帧数和帧间隔。

1. **返回绘图对象和动画对象**：

|  |
| --- |
| * return plt, anim |

* 返回Matplotlib的绘图对象和动画对象，以便进一步操作或展示。

在 **draw\_all\_charts 函数**中，核心功能是创建一个包含所有排序算法动态图表的复合图表。

以下是关键代码段及其说明：

1. **初始化图表和子图**：

|  |
| --- |
| * fig = plt.figure(1, figsize=(16, 9)) for i in range(6):  axs.append(fig.add\_subplot(331 + i)) |

* 创建一个新的图表，并为六种排序算法设置子图。

1. **隐藏坐标轴**：

|  |
| --- |
| * axs[-1].set\_xticks([]) axs[-1].set\_yticks([]) |

* 隐藏所有子图的坐标轴刻度，以便专注于排序动画。

1. **调整子图布局**：

|  |
| --- |
| * plt.subplots\_adjust(left=0.01, bottom=0.02, right=0.99, top=0.95, wspace=0.05, hspace=0.15) |

* 在一个图表中展示多种排序算法，每种算法一个子图，方便比较它们的效率和排序过程。
* 调整子图的位置和间距，确保它们在图表中的布局整洁且舒适。

1. **获取排序算法的动画帧**：

|  |
| --- |
| * for i in range(6):  frames.append(funs[i](data\_set)) |

* 为每种排序算法收集动画帧，每个动画帧代表排序过程中的一个步骤。

1. **打印排序算法信息**：

|  |
| --- |
| * for i in range(6):  print("%-\*s %\*d frames" % (max\_name\_length, names[i], max\_frame\_length, frame\_counts[i])) |

* 提供用户友好的反馈，告知用户每种排序算法的名称和涉及的动画帧数。

1. **定义动画函数**：

|  |
| --- |
| * def animate(fi):  # ... 动画绘制代码 ... |

* 对于每种排序算法，绘制当前帧的柱状图。

1. **创建动画对象**：

|  |
| --- |
| * anim = animation.FuncAnimation(fig, animate, frames=max(len(f) for f in frames), interval=frame\_interval) |

* 使用 FuncAnimation 连续播放所有排序算法的动画，允许用户同时观察所有算法的执行过程。

1. **返回绘图对象和动画对象**：

|  |
| --- |
| * return plt, anim |

* 返回Matplotlib的绘图对象和动画对象。

### 3.1.2 排序动画

**实现步骤：**

1. 初始化动画帧列表 frames。
2. 复制数据集 ds 以避免直接修改原始数据。
3. 通过两层循环实现冒泡排序，外层控制排序轮数，内层执行相邻元素的比较和交换。
4. 每次交换后，添加当前状态到动画帧列表，并标记交换元素。
5. 如果一轮排序未发生交换，提前结束排序，因为数据已经有序。
6. 排序完成后，将最终排序状态添加到动画帧列表。
7. 返回动画帧列表，供动画展示使用。

3.2程序运行

运行如下命令调用所有函数:

|  |
| --- |
| python output.py arg1 [arg2 [arg3]] |

**参数详情：**

参数1

* play：在新窗口中播放特定排序算法或所有算法的动画，作为Matplotlib的“图形”。
* save-html：将动画保存为一系列图像的HTML页面。
* save-mp4：将动画保存为MP4视频。

参数2

* all (默认)：在动画中显示所有排序算法的可视化。
* bubble-sort：仅在动画中显示冒泡排序算法的可视化。
* heap-sort: 仅在动画中显示堆排序算法的可视化。
* merge-sort: 仅在动画中显示归并排序算法的可视化。
* insertion-sort: 仅在动画中显示直接插入排序算法的可视化。
* selection-sort: 仅在动画中显示选择排序算法的可视化。
* quick-sort: 仅在动画中显示快速排序算法的可视化。

参数3

* random (默认)：对随机序列进行排序。
* almost-sorted：对几乎已排序的序列进行排序。
* few-unique：对一些部分重复的序列进行排序。
* reversed：对降序序列进行排序。

**示例**

要创建一个新窗口播放堆排序算法动画，初始序列为降序排列，可以运行：

|  |
| --- |
| python output.py play heap-sort reversed |

**运行截图**

* 冒泡算法对降序序列排序

|  |
| --- |
| python output.py play bubble-sort reversed |

|  |  |
| --- | --- |
| 排序前 | 排序后 |
|  |  |

* 堆排序算法对随机序列排序

|  |
| --- |
| python output.py play heap-sort |

|  |  |
| --- | --- |
| 排序前 | 排序后 |
|  |  |

* 归并排序算法对一些部分重复的序列排序

|  |
| --- |
| python output.py play merge-sort few-unique |

|  |  |
| --- | --- |
| 排序前 | 排序后 |
|  |  |

* 直接插入排序算法对几乎已排序的序列排序

|  |
| --- |
| python output.py play insertion-sort almost-sorted |

|  |  |
| --- | --- |
| 排序前 | 排序后 |
|  |  |

* 所有排序算法对随机的序列排序

|  |
| --- |
| python output.py play |

|  |  |
| --- | --- |
| 排序前 | 排序后 |
|  |  |

* 所有排序算法对随机的序列排序生成HTML

|  |
| --- |
| python output.py play |



4.测试与分析

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法名称 | 时间复杂度（平均） | 时间复杂度（最坏） | 空间复杂度 | 稳定性 | 适用场景 | 示例数据集表现 |
| 冒泡排序 | O(n^2) | O(n^2) | O(1) | 稳定 | 小规模数据或部分排序 | 慢 |
| 堆排序 | O(n log n) | O(n log n) | O(1) | 不稳定 | 大规模数据集 | 快 |
| 归并排序 | O(n log n) | O(n log n) | O(n) | 稳定 | 适合大数据量，可并行处理 | 快 |
| 直接插入排序 | O(n^2) | O(n^2) | O(1) | 稳定 | 小规模数据集或初始有序数据 | 较慢 |
| 选择排序 | O(n^2) | O(n^2) | O(1) | 不稳定 | 实时添加元素的场景 | 慢 |
| 快速排序 | O(n log n) | O(n^2) | O(log n) | 不稳定 | 大规模数据集 | 快（平均情况） |

排序算法的选择取决于数据规模、内存限制以及是否需要算法稳定性。对于小数据集，简单算法如冒泡排序或直接插入排序足够有效。对于大数据集，更高效的算法如堆排序、归并排序和快速排序是更好的选择。归并排序虽然内存需求较高，但提供了稳定性和并行处理能力。快速排序在平均情况下速度最快，但在最坏情况下性能会下降。总体而言，算法的选择需要平衡时间效率、空间消耗和稳定性。

5. 结论与心得

在这次课程设计中，我深刻体会到了理论与实践相结合的重要性。项目的核心是开发一个排序算法可视化工具，这不仅让我对冒泡排序、堆排序等经典算法有了更深入的理解，也锻炼了我的编程实践能力。在实现过程中，我遇到了不少挑战，比如动画的流畅性、性能优化以及用户交互设计等。为了解决这些问题，我查阅了大量资料，学习了新的技术和方法，这个过程虽然充满挑战，但也让我获得了巨大的成就感。

我学会了如何合理规划时间，将大目标分解为小任务，并一步步地完成它们。在调试代码时，我意识到了代码规范和模块化设计的重要性，这有助于快速定位并解决问题。

尽管项目中有许多困难，但通过不懈努力，我克服了这些障碍，并从中学到了很多。例如，我学会了如何使用Matplotlib库来制作动画，以及如何通过命令行参数让用户自定义排序算法和数据类型。这些技能对我未来的学习和工作都将产生积极的影响。

遗憾的是，由于时间限制，项目中仍有一些想法未能实现，比如增加更多排序算法的展示，以及进一步优化用户界面。这些未完成的部分也给了我未来努力的方向。我计划在接下来的时间里，继续完善这个项目，探索更多的功能和优化。

总的来说，这次课程设计是一次宝贵的学习经历。它不仅提升了我的技术能力，还锻炼了我的问题解决能力和创新思维。我相信，通过这次项目的经验，我将更加自信地面对将来的挑战，并不断追求卓越。

6.参考资料

[1] Cormen, T.H., Leiserson, C.E., Rivest, R.L. & Stein, C. Introduction to Algorithms [M]. Cambridge: MIT Press, 2009.

[2] Matthes, E. Python Crash Course [M]. Independently published, 2016.

[3] McKinney, W. Python for Data Analysis [M]. O’Reilly Media, 2017.