

《数据结构与算法设计》

课程设计总结

2021 年 7 月

目 录

第一部分 算法实现设计说明.....	1
1.1 题目	
1.2 软件功能	
1.3 设计思想	
1.4 逻辑结构与物理结构	
1.5 开发平台	
1.6 系统的运行结果分析说明	
1.7 操作说明	
第二部分 综合应用设计说明.....	
2.1 题目	
2.2 软件功能	
2.3 设计思想	
2.4 逻辑结构与物理结构	
2.5 开发平台	
2.6 系统的运行结果分析说明	
2.7 操作说明	
第三部分 实践总结.....	
3.1. 所做的工作.....	
3.2. 总结与收获.....	
第四部分 参考文献.....	

第一部分 算法实现设计说明

1.1 题目

堆的建立和筛选

输入一组关键值，用堆排序的方法进行从小到大的排序。

要求：(1)可以实现从小到大的排序，输出并显示该结果；

(2)随时显示输出堆顶元素，进行重新筛选后的堆。

1.2 软件功能

需根据题目的要求，详细设计相应软件的功能，以及这些功能的实现方式。

1. 输入一组数字。
2. 可视化该数字对应的堆。
3. 对该组数字进行堆排序，现实从小到大的结果。
4. 对堆排序过程中的每一次调整都进行可视化。
5. 可视化显示堆排序过程中被输出的堆顶元素。
5. 可以反复进行排序。

1.3 设计思想

需根据题目的要求，详细阐述软件的实现思路，算法设计基本流程等。

堆排序算法设计思路：

堆排序的输入是一份数组，而后依据从上到下，从左到右的顺序将数组的数字以此放到完全二叉树中，而后进行调整。

其中关键的一步调整为最大堆调整，函数如下：

```
function Max_Heapify(tree, start, end) {  
    var parent = start;  
    var child = parent * 2 + 1;  
    while (child <= end) { //防止出界  
        if (child + 1 <= end && get_real_number(tree[child]) < get_real_number(tree[child + 1])) //使得左右儿子中最大的值在child中  
        {  
            child += 1;  
        }  
        if (get_real_number(tree[parent]) < get_real_number(tree[child])) //父节点小于子节点则交换值  
        {  
            var t = tree[parent];  
            tree[parent] = tree[child];  
            tree[child] = t;  
            parent = child;  
            child = parent * 2 + 1;  
        } else { //交换完毕  
            break;  
        }  
    }  
    return tree;  
};
```

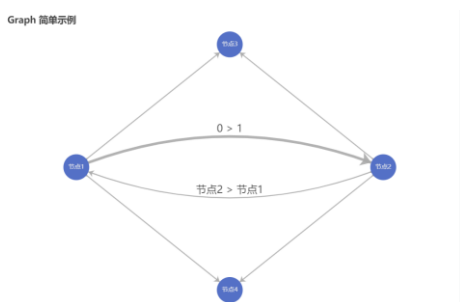
将 `start` 作为父节点，然后与左右 2 个子节点（如果有的话）中的最大值比较，如果父节点更大，那么就结束，而如果子节点更大，那么就交换父节点和该子节点的值，而后将该子节点当做父节点，继续向下看，直到父节点的值大于 2 个子节点的值或者达到叶子节点。

而后在堆排序的过程中，首先从最后一个非叶子节点开始，对其进行最大堆调整，而后继续往前，直到将前面所有的节点都调整完毕。这样就建立了一个最大堆，最大值在堆顶。而后将这个在堆顶的最大值与数组的最后一个值进行交换，同时将整个数组的需要遍历的长度-1，即将这个最大值排除在了数组之外，再对新交换来的堆顶元素进行最大堆调整，因为调整完后父节点总是大于 2 个子节点，故调整完毕后最大值还是在堆顶，而后进行同样的交换操作，直到需要遍历的长度为 0。这样数组中就是按照从小到大排序的数值了。而为了后续的可视化，我们还记录每一步调整的结果，放入到函数如下：

```
function heap_sort(tree) {
    var used_tree = []; //每一步遍历的数组
    var used_head = []; //每一步遍历后排除的
    var first = Math.floor(tree.length / 2) - 1; //得到第一个非子节点坐标
    for (var i = first; i >= 0; i--) {
        tree = Max_Heapify(tree, i, tree.length - 1)
    };
    used_tree.push(deepClone(tree));
    used_head.push([]);
    for (var j = tree.length - 1; j >= 0; j--) {
        var t = tree[0]; //交换堆顶和最后一个
        tree[0] = tree[j];
        tree[j] = t;
        tree = Max_Heapify(tree, 0, j - 1);
        var tt = [];
        var th = [];
        for (var i = 0; i < tree.length; i++) { //将结果记录
            if (i < j) {
                tt.push(tree[i]);
            } else {
                th.push(tree[i]);
            }
        }
        used_tree.push(deepClone(tt));
        used_head.push(deepClone(th));
    };
    var res = [];
    res.push(used_tree); //统一放入res中返回
    res.push(used_head);
    return res;
};
```

可视化堆的思路

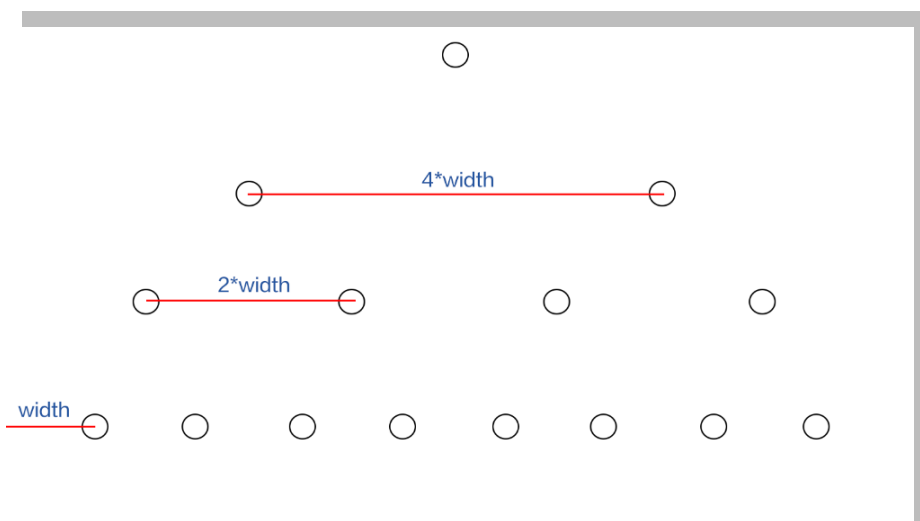
这里使用的可视化工具为 [Apache Echarts](#) 库中的关系图部分。它可以通过 js 控制，很方便地在网页中实现可视化。关系图的效果如下：



最基础的，我们只需要给定节点的 `name`、`x`、`y` 还有连线两端的节点的 `name` 就可以得到一幅简单的关系图，而后可以对节点的大小、颜色等等进行调整。

可视化堆中最先需要解决的是各个节点的 `x`、`y` 坐标如何确定的问题。解决思路如下：

首先得到该堆对应的完全二叉树的深度 `depth`，然后得到最低一层对应的填满后的节点个数，即 $2^{\text{depth}-1}$ ，而后根据画布的宽度 `wide`，就可以得到最底下相邻 2 个节点之间的距离 $\text{width} = \text{wide} / (2^{\text{depth}-1} + 1)$ 。而这上一层的相邻的宽度为 $2 * \text{width}$ ，再往上一层是 $2^2 * \text{width}$ ，以此类推，如下：



而后再观察得到最左边的节点的 `x` 坐标=下一层最左边节点的 `x` 坐标+下一层相邻 2 节点距离/2，而后得到的表达式如下：

$$x = 2^{\text{depth} - \text{now_depth} - 1} * \text{width} + \text{width} / 2$$

而其他点就依据最左边的坐标依次加上对应的距离即可。

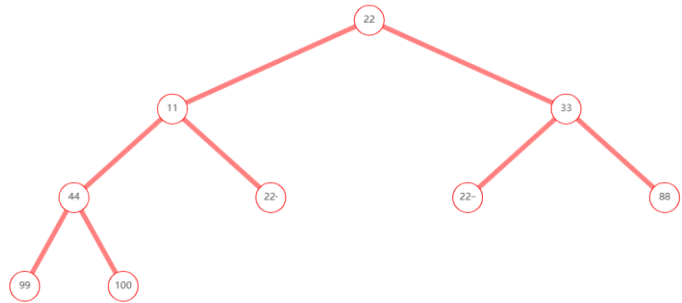
而对于 `y` 坐标只需要在得到该节点对应的深度后固定乘宽度即可。

对于 `name` 的赋值因为存在多个数值相同的情况，而各个节点的 `name` 又必须为唯一的，固这里采取了一种折中的方案，对于相同的数字，在其后面加上不同的 `•` 来作为区别。

而后需要确定连线的头尾的 `name`，这里就依据父节点和左右子节点的坐标规律即可，

假设父节点的坐标为 i （从 1 开始），则其左儿子节点的坐标为 $2*i$ ，右儿子节点为 $2*i+1$ 。
这样就得到了对应的连线的数据。

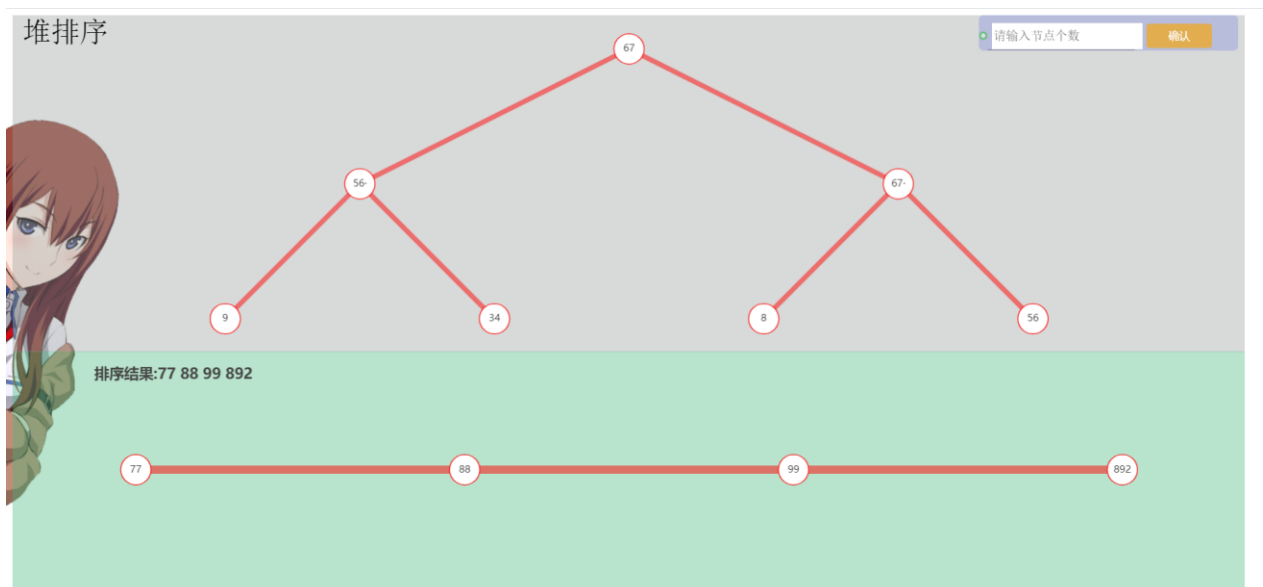
对于数组 [22, 11, 33, 44, 22, 22, 88, 99, 100] 可视化的结果如下：



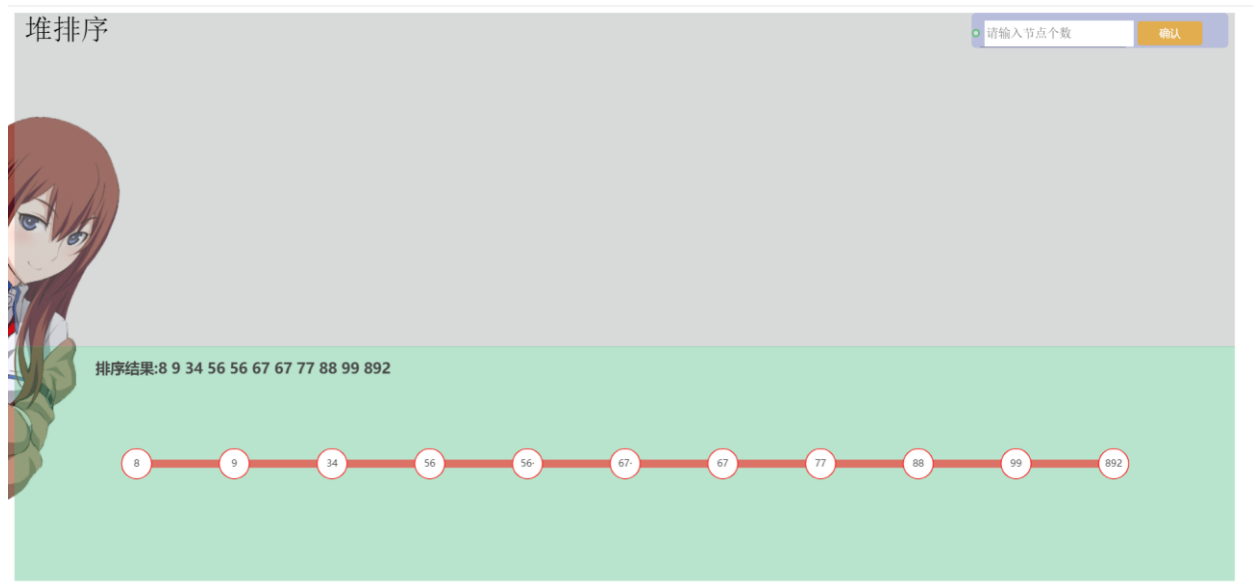
（可以看到这里存在多个 22，而节点中的 22 后面跟上了不同数量的 • 用以区分）

动态展示堆排序过程

根据前面堆排序的说明可以知道我们在堆排序过程中已经将过程中的数据通过数组的形式保存了下来，对于堆的展示，我们只需要将那些数组依次传递到可视化堆的函数之中，而对于保存的每一步排序完毕的记录下的数组，我们将其在另一个地方放置展示即可，得到的效果如下：



灰色部分为正在排序的堆，绿色部分以及排序好的每次取出的堆顶元素，最后得到的效果如下：



1.4 逻辑结构与物理结构

完整写出程序中应用的逻辑结构及物理结构。

整个程序分为 html 部分、css 部分、js 部分。

html 部分

主要是初始化界面时的骨干部分，以及便于后续 js 改变界面做定位。

内容截图如下：

```
<!DOCTYPE html>
<html lang="CN">

<head>
  <meta charset="UTF-8">
  <script src="/static/echart_5.1.js"></script>
  <link href="dui.css" rel="stylesheet" type="text/css" />
  <title>堆排序</title>
</head>

<body>
  <div id='tip'>堆排序</div>
  <div id='bj1'>§</div>
  <div id="info">
    
    <input type="text" id="sum" placeholder="请输入节点个数" onkeyup="if(isNaN(value))execCommand('undo');" onafterpaste="if(isNaN(value))execCommand('undo')">
    <input type="button" onclick="add_number_k()" id="sure" value="确认">
    <div id="start"></div>
  </div>
  <div id="main"></div>
  <div id="res"></div>
  <script src="dui_change.js">
  </script>
  <script src="dui.js">
  </script>
</body>
</html>
```

CSS 部分

主要用于美化布局，规定了各个部分的大小、颜色和位置。

这里主体的美化工程为：

- 1.用于展示堆的 div
- 2.用于展示被排出的堆顶元素的 div
- 3.用于输入数据的文本框和按钮

4.背景

部分截图如下：

```
#infor {
    position: fixed;
    height: auto;
    z-index: 0;
    overflow-y: auto;
    right: 2%;
    background-color: #afb4bc6;
    max-height: 300px;
    border-radius: 6px;
}

#main {
    height: 57%;
    width: 98%;
    margin: 0;
    padding: 0;
    position: absolute;
    z-index: -2;
    background-color: #999d9c61;
    padding: 0px;
}

#res {
    height: 40%;
    width: 98%;
    margin: 0;
    padding: 0;
    position: absolute;
    top: 58%;
    border: 0px;
    z-index: -2;
    background-color: #45b97b61;
}

#sum {
    outline: 0;
    border: 0px solid #000000;
    width: 180px;
    height: 30px;
    font-size: 15px;
}
```

js 部分

js 部分是该程序的核心所在，主要分为 3 个部分：修改输入数据的文本框、对数据进行堆排序、对堆排序结果的展示。

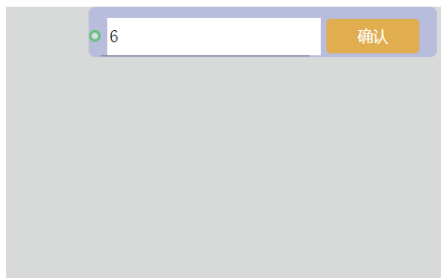
1. 修改输入数据的文本框

这部分主要是需要先得到输入节点的个数，而后在该 div 中插入 HTML 代码，加入输入数据的文本框和确认排序按钮。

初始的输入界面部分如下：



而后输入节点个数，点击确认。



得到如下的界面：



将对应数值输入后点击开始排序，又将该部分删除，同时将输入节点个数的那个文本框内的数字清空，得到如下：



主要依靠的是 `document.getElementsByClassName()`和 `document.getElementById` 进行定

位然后依据 insertAdjacentHTML()进行插入代码，依据 remove()删除对应的代码。代码截图如下：

```
function add_number_k() {
    var sum = parseInt(document.getElementById("sum").value, 10);
    var start = document.getElementById("start");
    var add_txt = '\n\n<input type="text" class="number" placeholder="请输入数值" onkeyup="if(isNaN(value))execCommand(\`undo\`)" onafterpaste="if(isNaN(value))execCommand(\`undo\`)">\n\n<div class="fengge"></div>';
    if (sum == '') {
        alert('无数字');
    } else {
        for (var i = 0; i < sum; i++) {
            start.insertAdjacentHTML('AfterEnd', add_txt);
        }
    }
    var t = document.getElementsByClassName("fengge");
    t[t.length - 1].insertAdjacentHTML('AfterEnd', '<input type="button" onclick="dui_draw()" id="sousuo" value="开始排序">');
    var ts = document.getElementById("sousuo");
    ts.insertAdjacentHTML('AfterEnd', '<input type="text" id="time_set" placeholder="动画停留时间，默认2000ms" onkeyup="if(isNaN(value))execCommand(\`undo\`)" onafterpaste="if(isNaN(value))execCommand(\`undo\`)">');
}

function close_win() {
    var n = document.getElementsByClassName("number");
    var f = document.getElementsByClassName("fengge");
    var img = document.getElementsByClassName("imglv");
    var ss = document.getElementById("sousuo");
    var ts = document.getElementById("time_set");
    ss.remove();
    ts.remove();
    var l = n.length;
    for (var i = 0; i < l; i++) {
        n[i].remove();
        f[i].remove();
        img[i].remove();
    }
    var sum = document.getElementById("sum");
    sum.value = '';
}
```

2. 对数据进行堆排序

对数据堆排序的部分见前面堆排序算法部分即可。

3. 对堆排序结果的展示

这部分主要是分为 2 个模块，一个是画堆，还有一个是画每次排出的堆顶元素的图。主要介绍见前面可视化部分。

1.5 开发平台

说明所采用的开发平台，如果采用了第三程序包或者开源代码，请详细说明。

同时说明软件的运行环境。

开发平台：Visual Studio Code

ECharts 开源代码：echart_5_1.js

介绍：ECharts 是一款基于 JavaScript 的数据可视化图表库，提供直观，生动，可交互，可个性化定制的数据可视化图表。ECharts 最初由百度团队开源，并于 2018 年初捐赠给 Apache 基金会，成为 ASF 孵化级项目。2021 年 1 月 26 日晚，Apache 基金会官方宣布 ECharts 项目正式毕业。1 月 28 日，ECharts 5 线上发布会举行。

软件运行环境：Windows 10 Chrome 浏览器

1.6 系统的运行结果分析说明

简要说明应用开发工具进行调试及开发的过程。

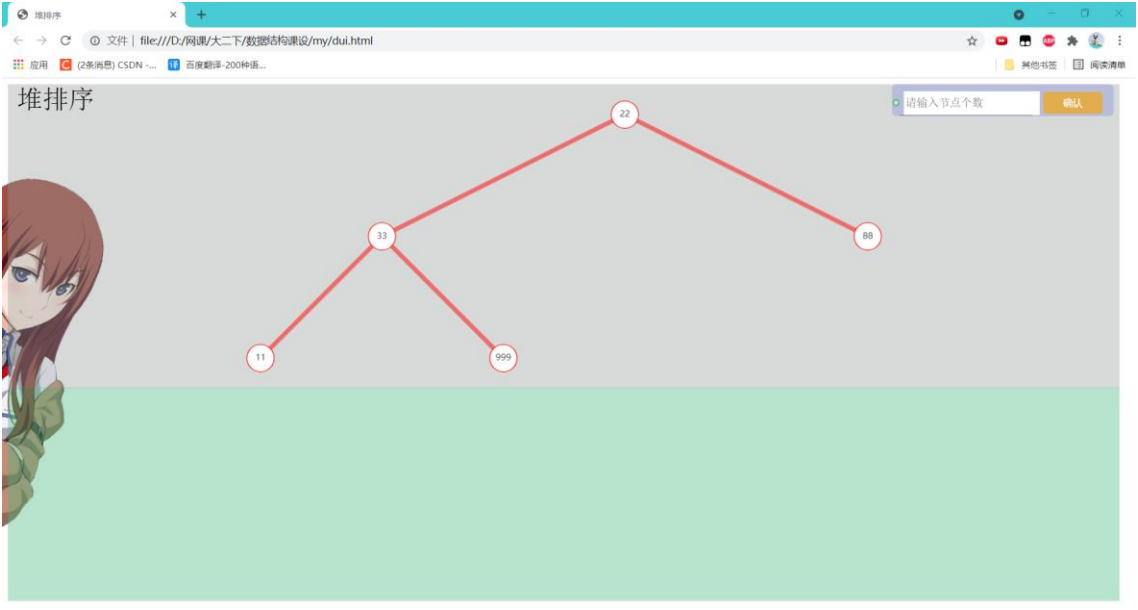
调试主要是依靠 Visual Studio Code 的插件[Deprecated] Debugger for Chrome，从而可以对 js 文件进行分步调试和对变量进行监视。

运行结果经过检验均正确，稳定性较好、具有一定的容错能力。

运行案例：

输入数据：[22, 33, 88, 11, 999]

初始堆：

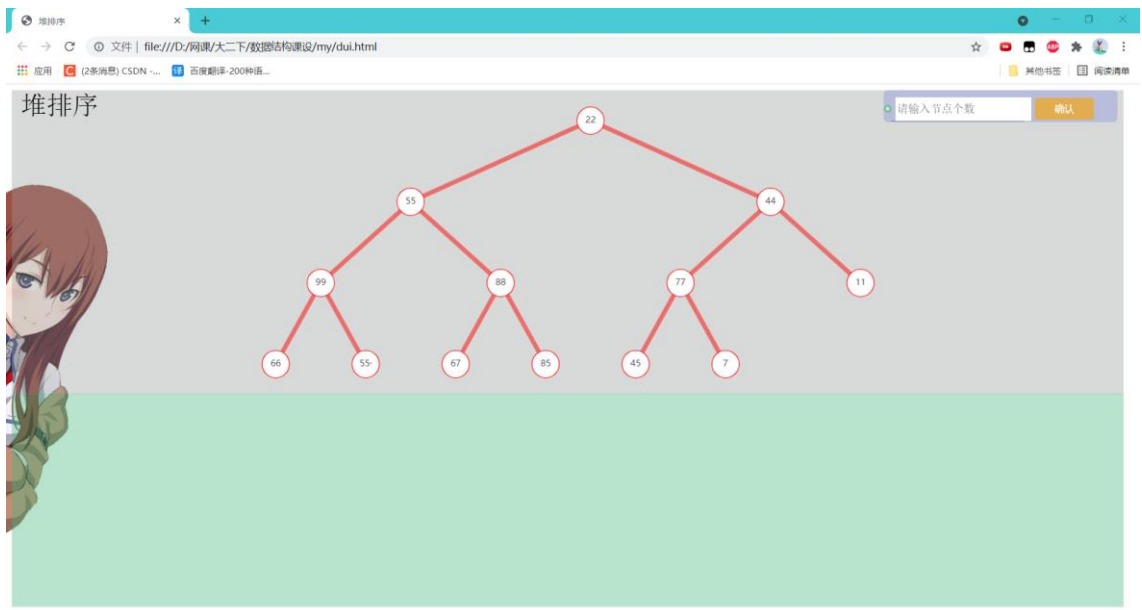


排序结果



输入数据：[22, 55, 44, 99, 88, 77, 11, 66, 55, 67, 85, 45, 7]

初始堆：



排序结果：



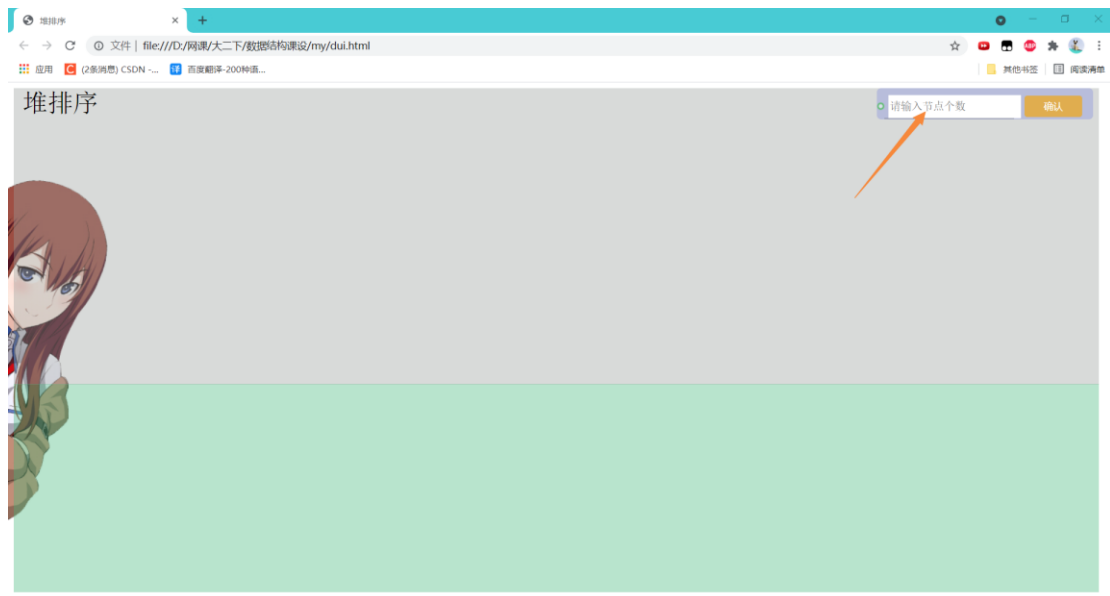
可知均为从小到大排序，结果正确

说明所开发软件达到的成果，包括运行的正确性、稳定性、容错能力等。

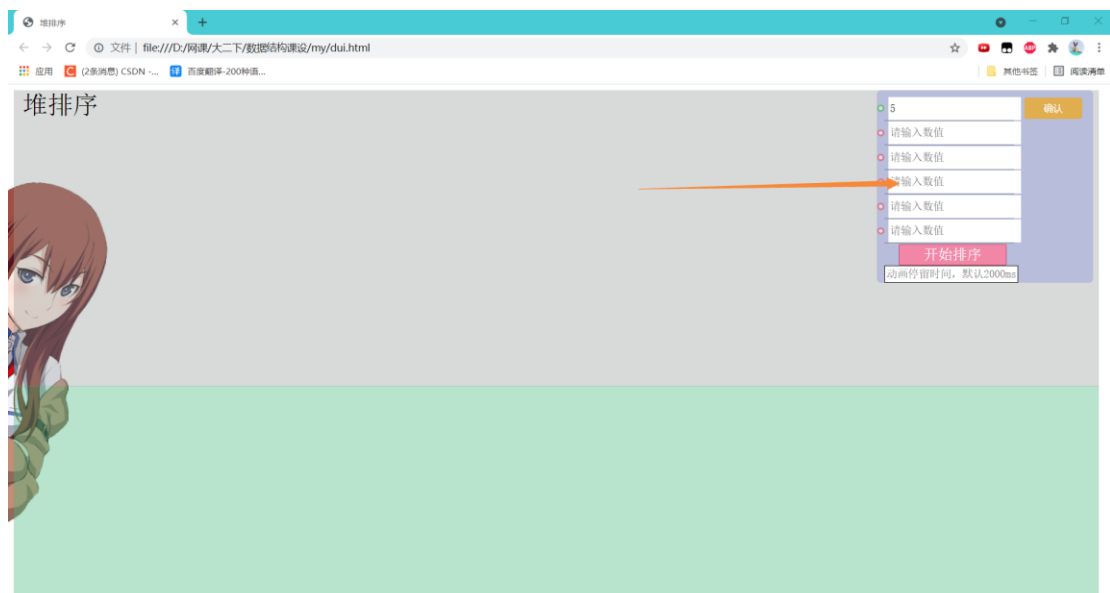
采用运行案例的方式说明运行结果，在报告中给出具体的案例，可以是多组案例分别体现不同的运行状态。

1.7 操作说明

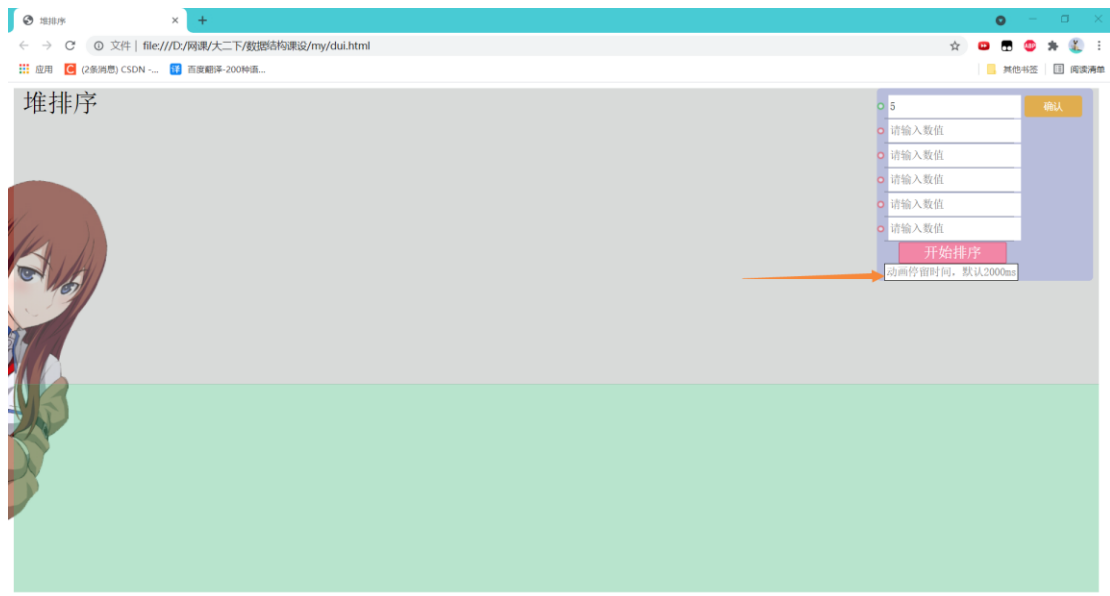
1. 输入节点个数



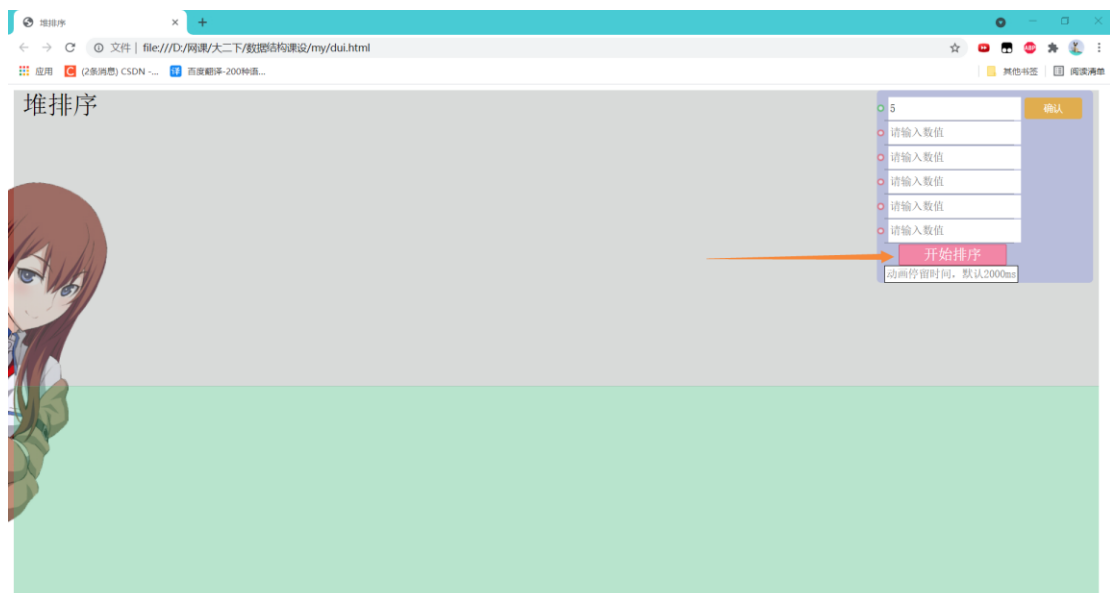
2. 输入节点数据



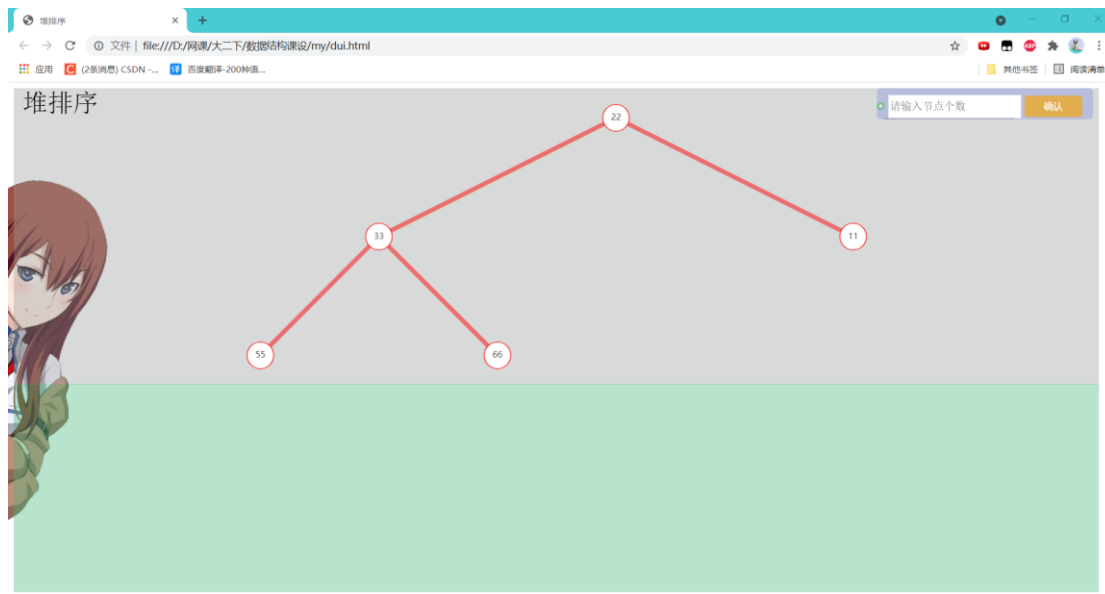
3. 如果有需要输入每一次动画停留的时间，若不输入则默认 2s



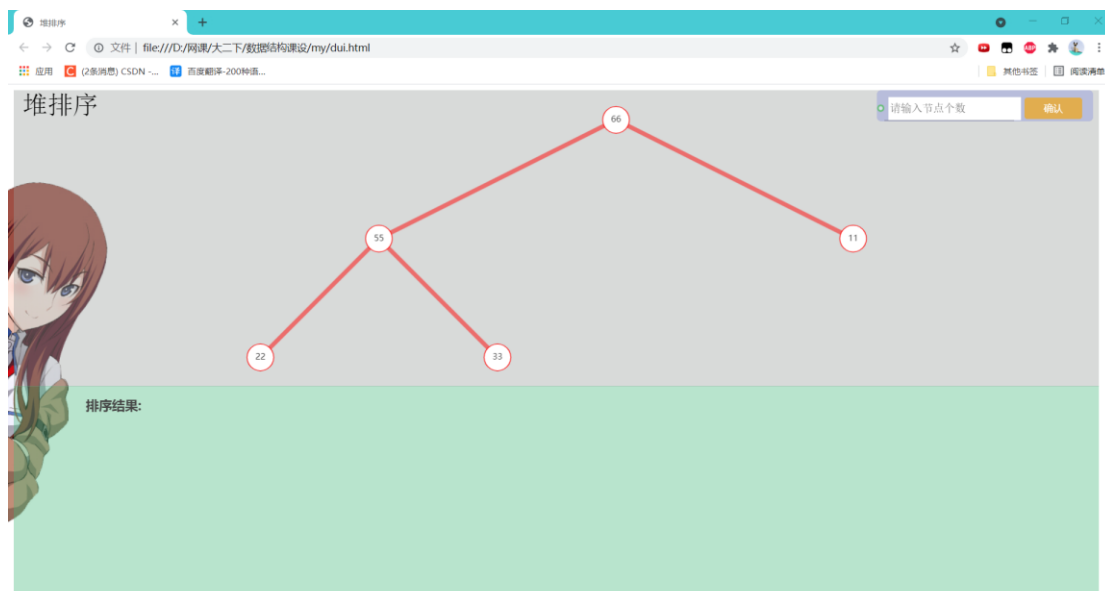
3. 点击开始排序按钮，进行排序



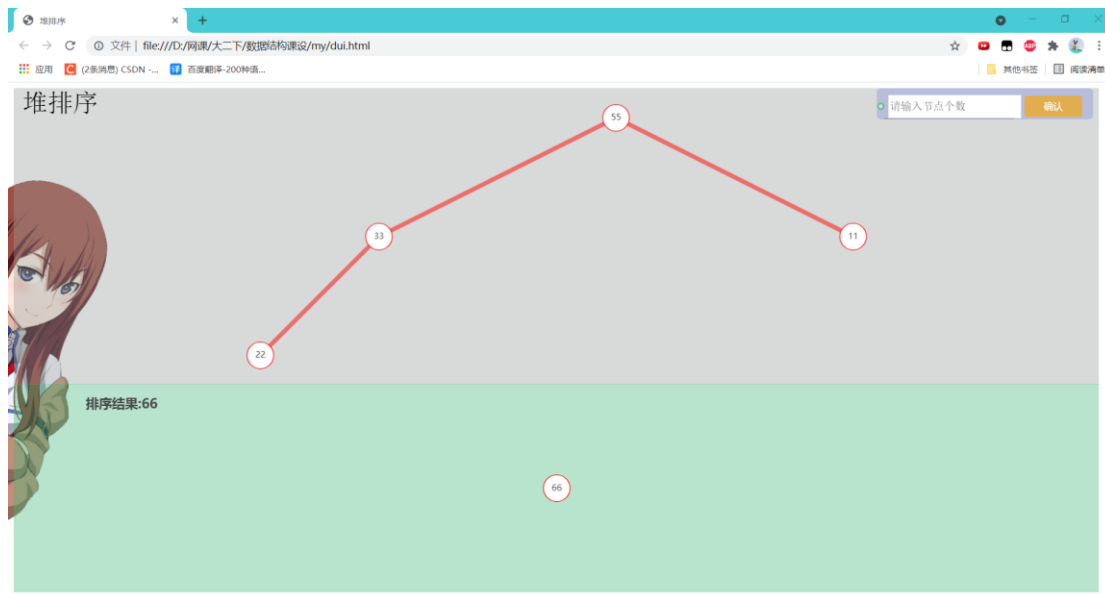
4. 观察等待排序结果



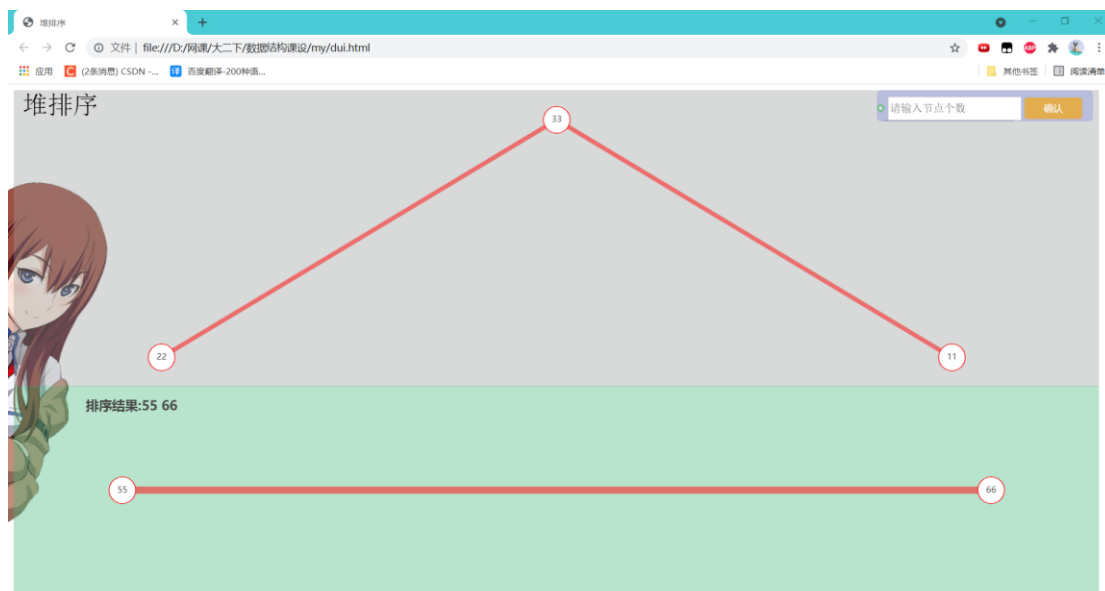
（这是依据输入数据的顺序排列的完全二叉树）



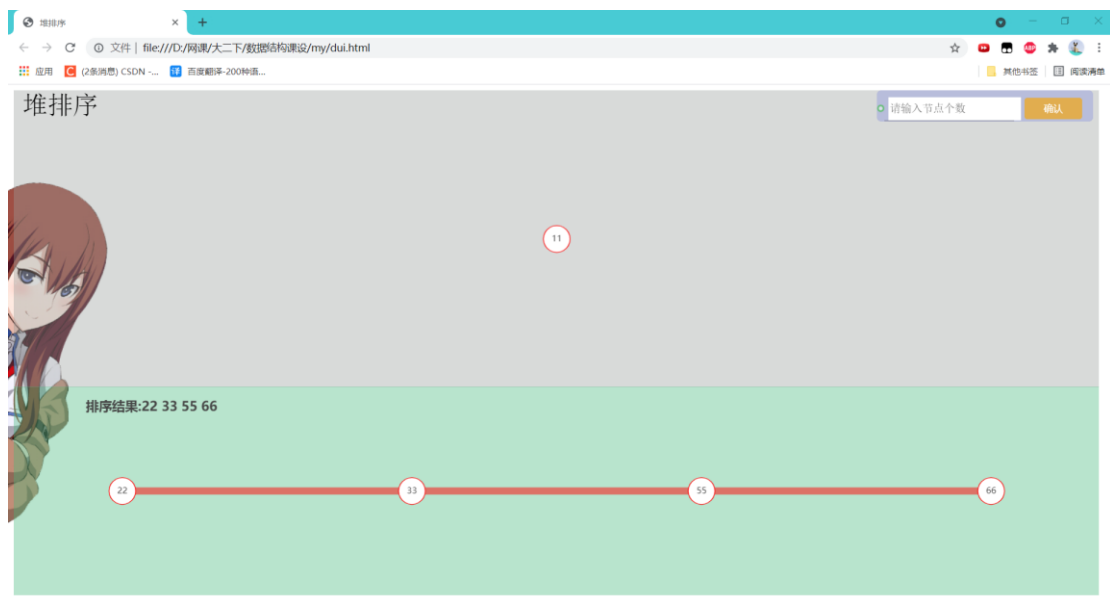
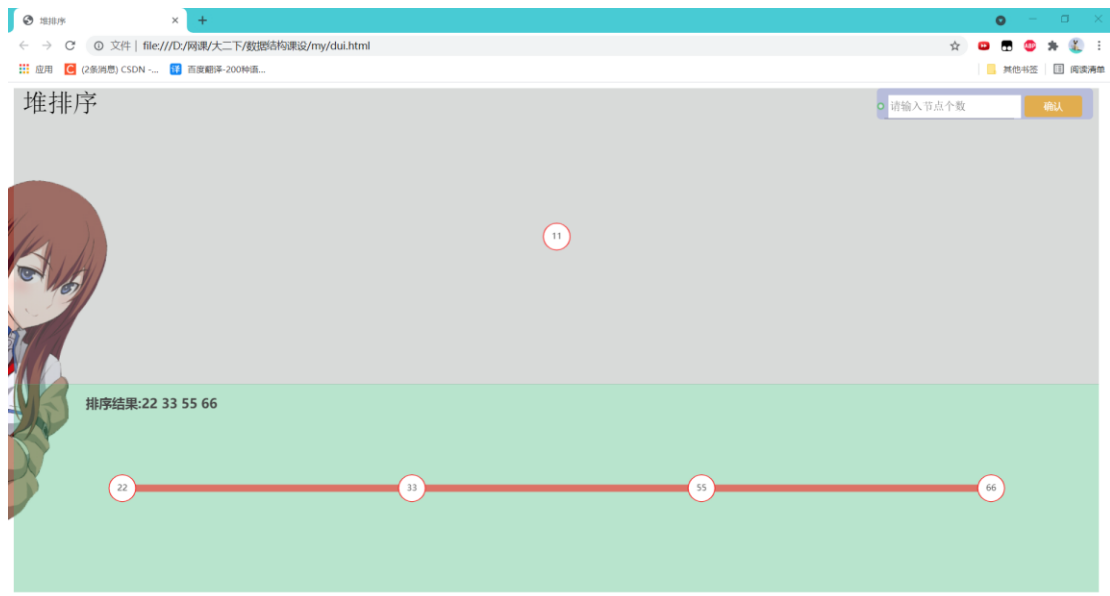
（进行了一次从下往上堆调整的结果）



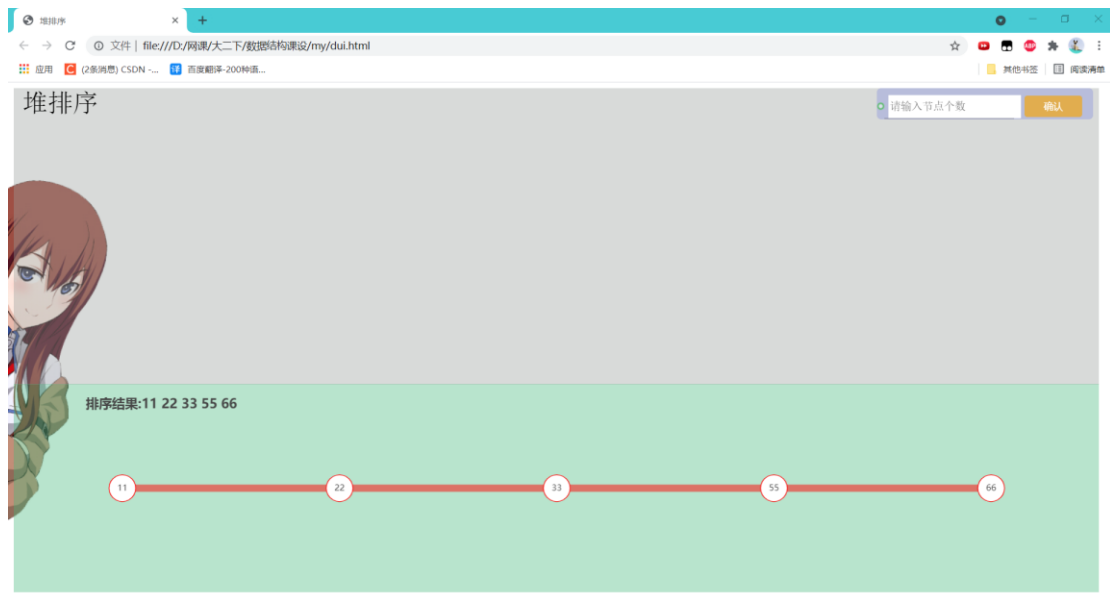
（堆顶元素被取出）



（堆顶元素被取出）



（堆顶元素被取出）



（堆顶元素被取出，完成排序）