

浙江大学

**Data Structure & Database Technique**  
**Project4**  
**Project Report**



2020~2021 春夏学期    2021    年   4   月   5   日

## Categories

<b>CHAPTER 1: INTRODUCTION .....</b>	<b>3</b>
<b>CHAPTER 2: ALGORITHM SPECIFICATION.....</b>	<b>3</b>
链表排序:.....	3
链表排序方法 .....	3
ARRAYLIST 排序: .....	4
<b>CHAPTER 3: TESTING RESULTS .....</b>	<b>4</b>
<b>CHAPTER 4: ANALYSIS AND COMMENTS .....</b>	<b>6</b>

# Project4

## Chapter 1: Introduction

Background and Our goals :

对于之前实现的链表和 `ArrayList`，分别添加一个函数 `sort`

- 能够对线性表中存储的数据进行排序
- 函数要有单元测试
- 思考：哪些排序方法适合链表

## Chapter 2: Algorithm Specification

### 链表排序:

双指针法实现快速排序,

**I = 0, j = i+1**, 设 **i** 为基准数字.

**While(j < nums.size):**

- 如果 **j** 指向的值大于等于基准数字（如果比基准大，直接跳过）
  - **j ++**
- 如果 **j** 指向的值小于基准数字，（如果比基准小，交换 **i** 和 **j** 位置的值）
  - **i ++**
  - **swap(i, j)**
  - **j ++**

这样就把基准放在中间，左边都是比基准小右边比基准大.然后递归即可。

### 链表排序方法

可以插入排序，时间复杂度  $O(n^2)$  ,空间复杂度  $O(1)$

选择排序（时间复杂度  $O(n^2)$  ,空间复杂度  $O(1)$ ）

快速排序（平均时间复杂度  $O(n \log n)$  ,不考虑递归栈空间的话空间复杂度是  $O(1)$ ）

归并排序（算法交换链表节点，时间复杂度  $O(n \log n)$  ,不考虑递归栈空间的话空间复杂度是  $O(1)$ ）

首先用快慢指针的方法找到链表中间节点，然后递归的对两个子链表排序，把两个排好序的子链表合并成一条有序的链表。归并排序应该算是链表排序最佳的选择了，保证了最好和最坏时间复杂度都是

$n \log n$ ，而且它在数组排序中较大的空间复杂度在链表排序中也从  $O(n)$  降到了  $O(1)$

**冒泡排序**（算法交换链表节点 `val` 值，时间复杂度  $O(n^2)$ ，空间复杂度  $O(1)$ ）

对于**希尔排序**，因为排序过程中经常涉及到 `arr[i+gap]` 操作，其中 `gap` 为希尔排序的当前步长，这种操作不适合链表。

对于**堆排序**，一般是用数组来实现二叉堆，当然可以用二叉树来实现，但是这么做太麻烦，还得花费额外的空间构建二叉树

## Arraylist 排序:

ArrayList 的好处是可以不用限定容器的大小，他会根据元素的增加自己扩大。但是存储进去的数据类型都会变成 `object`，虽然每个元素有自己的 `index`，但不像数组的下标可以更加方便的操作. 最开始的笨办法就是把 `list` 中的数据传给数组排序好了再传回来。但是这样效率下降过多。

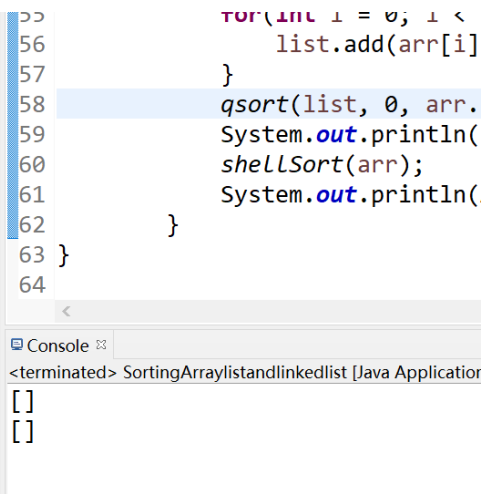
Shell 排序

- 时间复杂度的下界是  $n \log_2 n$
- 空间复杂度为  $O(1)$
- 由于多次插入排序，我们知道一次插入排序是稳定的，不会改变相同元素的相对顺序，但在不同的插入排序过程中，相同的元素可能在各自的插入排序中移动，最后其稳定性就会被打乱，所以 `shell` 排序是不稳定的。

## Chapter 3: Testing Results

test case	Correct answer	Actual behavior of my program
链表排序 <code>{5,4,3,2,1,6,7,9,8,10}</code>	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	<pre> 35 36 public static void main(String[] args) { 37     LinkedList&lt;Integer&gt; list = new LinkedList&lt;Integer&gt;(); 38 //     int[] arr = {4,2,5,3,7,9,0,1}; // test case 1 39     int[] arr = {5,4,3,2,1,6,7,9,8,10}; // test case 2 40     for(int i = 0; i &lt; arr.length; i++) { 41         list.add(arr[i]); </pre> <p>Console: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]</p>

<p>{4,2,5,3,7,9,0,1}链表排序</p>	<p>[0, 1, 2, 3, 4, 5, 7, 9]</p>	<pre> 37         LinkedList&lt;Integer&gt; list = new Link 38         int[] arr = {4,2,5,3,7,9,0,1}; // t 39 //         int[] arr = {5,4,3,2,1,6,7,9,8,10}; / 40         for(int i = 0; i &lt; arr.length; i++) 41             list.add(arr[i]); 42     } 43     qsort(list, 0, arr.length-1); 44     System.out.println(list.toString()) 45 } 46 } 47 </pre> <p>Console</p> <p>&lt;terminated&gt; SortingArraylistandlinkedlist [Java Application] C:\Program Files\Java\jdk</p> <p>[0, 1, 2, 3, 4, 5, 7, 9]</p>
<p>int[] arr = {};链表排序 空测试</p>	<p>[]</p>	<pre> 40         int[] arr = {}; 41         for(int i = 0; i &lt; arr.length; i++) 42             list.add(arr[i]); 43     } 44     qsort(list, 0, arr.length-1); </pre> <p>Console</p> <p>&lt;terminated&gt; SortingArraylistandlinkedlist [Java Application]</p> <p>[]</p>
<p>Shellsort 线性表</p>	<p>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]</p>	<pre> 57     } 58     qsort(list, 0, arr.length-1); 59     System.out.println(list.toString()); 60     shellSort(arr); 61     System.out.println(Arrays.toString(arr)); 62 } 63 } 64 </pre> <p>Console</p> <p>&lt;terminated&gt; SortingArraylistandlinkedlist [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (202</p> <p>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]</p> <p>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]</p>

<p>Shellsort 线性表 零输入</p>	<p>[]</p>	 <pre> 55         for(int i = 0; i &lt; 56             list.add(arr[i] 57         } 58         qsort(list, 0, arr. 59         System.out.println( 60             shellSort(arr); 61         System.out.println( 62             } 63     } 64 </pre> <p>Console</p> <p>&lt;terminated&gt; SortingArraylistandlinkedlist [Java Application</p> <p>[]</p> <p>[]</p>
------------------------------	-----------	--

## Chapter 4: Analysis and Comments

Quicksort time complexities is  $O(n \log n)$ , Each time it is exchanged, it will not be like bubble sorting and can only exchange between adjacent numbers each time, and the exchange distance is much larger. Therefore, the total number of comparisons and exchanges is reduced, and the speed is naturally increased. In the worst case, it may still be the exchange of two adjacent numbers. Therefore, the worst time complexity of quick sort and bubble sort is the same as  $O(N^2)$ , and its average time complexity is  $O(N \log N)$ .

space complexities of the algorithms

The all space complexities is  $O(1)$  , we don't need much extra space, just a few pointers and temporary variables