

数据结构与算法 作业报告

第二次



姓名 曹家豪

班级 软件 2204 班

学号 2226114017

电话 13572763245

Email caojiahao@stu.xjtu.edu.cn

日期 2023-12-23

目录

任务一：	2
1.题目： 为指定的 List ADT 实现各种数据结构.....	2
2.数据与算法设计	2
3.部分代码说明.....	4
4.运行结果：	11
5.总结收获：	13
任务二	13
1.题目： 创建一个可自动调整空间大小的 List 数据结构.....	13
2.数据与算法设计：	14
3.部分代码说明：	14
4.运行结果展示：	16
5.总结与收获：	17
任务三	17
1.题目： 栈.....	17
2.快速排序：	18
3.计算器：	20
4.LeakyStack：	23
5.总结与收获：	24
任务四	25
1.题目： 基数排序.....	25
2.数据与算法设计：	25
3.部分代码说明：	25
4.运行结果展示：	30
5.总结与收获：	34

任务一：

1.题目：为指定的 List ADT 实现各种数据结构

在本次实验中，主要完成的任务是：

1、为指定的 List ADT（该 ADT 的具体要求请参见文件 List.java）实现三种数据结构：

①使用顺序数组做为存储表示；

②使用单向链表做为存储表示；

③使用双向链表做为存储表示。

不论哪种存储表示下实现的数据结构，实验中需要处理的数据类型都是 Character 类型。

2、用给定的输入数据文件验证所实现的数据结构是否正确。

3、使用表格列举每种数据结构下所实现的所有方法的时间复杂度。

2.数据与算法设计

首先观察可以得到该 List ADT 接口中定义了 insert、remove、replace 等 14 种方法，要实现该 ADT，则三种数据结构均要实现这 14 种方法，其中 insert、remove、replace 等操作相对而言考虑因素较多，还需要注意的是，由于采取链式存储，后两种实现方式需要在数据成员中定义 Node 节点，并将 Character 以 Node 方式添加至链中。

下面针对不同存储方式对相关方法做出说明：

首先说明对三种存储方式不同的初始化，在顺序数组的初始化中，定义一个新数组，使 length=0，cursor=-1；在两种链式存储中，令头节点与尾节点指向 nul，cursor 指向头节点，length=0。以上也是 clear（）方法所要进行的操作。

其次下面对三种存储方式下具体实现有较大差异的 insert（）、remove（）、replace（）三种方法分别做出解释：

方法	文字描述	存储方式	实现逻辑
insert (T)	如果 list 已满则抛出自定义 ListExceptio 异常，否则将新元素 T 插入 cursor 指向元素的下一个位置，并将 cursor 指向新元素	顺序数组	数组已满时抛出异常，否则从 cursor 下一个位置开始直到 list 末尾，将数组每个位置元素挪至下一个位置。完成后将 T 放置在 cursor 的下一个位置并更新 cursor 和 length
		单向链表	链表为空则将 T 链至头节点，更新 cursor、length 与 tail，否则以 cursor→T→cursor.next 的形式更新单向链表及 length 值，如果 cursor 是链表末尾还需更新 tail
		双向链表	与单向链表操作逻辑大致相同，需要的额外操作是，更新链表节点时要同时更新节点的前接节点与后接节点，同时能够处理

		表	cursor 指向链表头和尾的边界情况
remove ()	在 list 不空的情况下，删除 cursor 指向的元素并将 cursor 指向被删除元素的下个位置，倘若被删除元素是 list 的末尾元素，则将 cursor 移至 list 开头（若 list 仅有 cursor 指向的唯一元素，则需要在删除后对 list 进行初始化）	顺序数组	数组为空时抛出异常，否则将从 cursor 开始至倒数第二个元素，每个位置用下一个位置的元素替换，更新 length。此时若 length=0 说明数组为空需要初始化 list，若 cursor=length 说明原本 cursor 指向元素为 list 末尾，此时要将 cursor 挪至数组开头
		单向链表	链表为空时抛出异常，否则需要通过遍历找到 cursor 元素的前接节点 temp，令 temp 的 next 指向 cursor.next，若此时链表为空则初始化链表，此外如果原本 cursor 指向链表末尾则需要更新 tail，并将 cursor 指向 head.next，否则更新 cursor 至 temp.next，最后更新 length
		双向链表	与单向链表操作逻辑类似，但 cursor 的前接节点可以直接获取而不需要遍历，更新链表要修改节点的前接与后接节点
replace (T)	将 cursor 指向位置元素替换为 T，当 list 为空时，插入 T 并更新 cursor	顺序数组	如果数组为空，则进行插入 T 操作，否则直接将数组中 cursor 指向元素改为 T
		单向链表	如果链表为空则进行插入 T 操作，否则仍需找到 cursor 的前接节点 temp，以 temp→T→cursor.next 形式更新链表令 cursor 指向 T，如果 cursor 原本指向链表末尾则更新 tail
		双向链表	与单向链表操作类似，更新链表要修改节点的前接与后接节点

此外因为在链式存储中 cursor 代表的是节点而非直接的数字序号，因此为获取链式存储中国 cursor 元素所处的位置序号，定义 getcursorlnedx 方法，初始化 index=-1，从头结点开始遍历，每经过一个节点就更新 index 的值，直至 cursor 元素，返回 index。

其他方法的实现逻辑相对简单，或是与以上方法均有相似之处（如 getPrev 获取节点的前接节点），不再详细说明。

主测试函数的编写：

由于规定了利用字符操作的方式，我们定义 execute 方法，传入操作字符串与 list，通过多个 switch 语句来判断要进行的操作，完成对 list 进行的操作。如何在主函数中，定义 list 对象指向不同数据结构，再依次读取测试文件的每一行存入 line，将 line 与 list 传入 execute 方法，美处理一行就调用 showStructure 方法输出，得到最终结果。

3.部分代码说明

Insert 方法的三种实现:

顺序数组	<pre> 1. public void insert(Character newElement) throws List Exception{ 2. if(isFull()){ 3. throw new ListException("List is full"); 4. // 数组满则抛出异常 5. } 6. else{ 7. for(int i=length;i>cursor+1;i--){ 8. array[i] = array[i-1]; 9. // 将后面的元素向后移动一位 10. } 11. array[++cursor] = newElement; 12. // 将新元素插入到cursor 的位置 13. length++; 14. } 15. }</pre>
单向链表	<pre> 1. public void insert(Character newElement) throws List Exception{ 2. // 检查链表是否已满 3. if(isFull()) { 4. // 如果已满，则抛出异常 5. throw new ListException("List is full"); 6. } else { 7. // 创建一个新节点 8. Node<Character> newNode = new Node<Character> >(newElement); 9. // 检查链表是否为空 10. if(isEmpty()) { 11. // 如果为空，将新节点设置为头节点、尾节点和 游标位置 12. head.setNext(newNode); 13. tail = newNode; 14. cursor = newNode; 15. } else { 16. // 如果不为空，将新节点的下一个节点设置为当 前游标位置的下一个节点 17. newNode.setNext(cursor.getNext()); 18. // 将当前游标位置的下一个节点设置为新节点 19. cursor.setNext(newNode); 20. // 将游标位置移动到新节点 21. cursor=cursor.getNext(); 22. // 检查游标位置的下一个节点是否为空 23. if(cursor.getNext()==null){ 24. // 如果为空，将尾节点设置为当前游标位置 25. tail=cursor; 26. } }</pre>

		<pre> 27. } 28. // 链表长度加一 29. length++; 30. } 31. }</pre>
双向链表		<pre> 1. public void insert(Character newElement) throws ListException{ 2. // 如果链表已满，抛出异常 3. if(isFull()) { 4. throw new ListException("List is full"); 5. } else { 6. // 创建新节点 7. Node<Character> newNode = new Node<Character>(newElement); 8. // 如果链表为空 9. if(isEmpty()) { 10. // 将新节点设为头节点、尾节点和游标位置 11. head.setNext(newNode); 12. tail = newNode; 13. cursor = newNode; 14. } 15. // 如果链表只有一个节点 16. else if(cursor==head){ 17. // 将新节点的下一个节点设为头节点的下一个节点 18. newNode.setNext(head.getNext()); 19. // 更新头节点的下一个节点的前驱节点 20. head.getNext().setPrev(newNode); 21. // 将头节点的下一个节点设为新节点 22. head.setNext(newNode); 23. // 将游标位置设为新节点 24. cursor=newNode; 25. } 26. // 如果链表只有一个节点 27. else if(cursor==tail){ 28. // 将当前游标位置的下一个节点的前驱节点设为新节点 29. newNode.setPrev(cursor); 30. // 将当前游标位置的下一个节点设为新节点 31. cursor.setNext(newNode); 32. // 将尾节点设为新节点 33. tail=newNode; 34. // 将游标位置设为新节点 35. cursor=newNode; 36. } 37. // 如果链表有多个节点 38. else { 39. // 将新节点的下一个节点设为当前游标位置的下一个节点 40. newNode.setNext(cursor.getNext()); 41. // 更新当前游标位置的下一个节点的前驱节点 42. cursor.getNext().setPrev(newNode);</pre>

	<pre> 43. // 将当前游标位置的前驱节点设为新节点 44. newNode.setPrev(cursor); 45. // 将当前游标位置的下一个节点设为新节点 46. cursor.setNext(newNode); 47. // 将游标位置移动到新节点的下一个节点 48. cursor=cursor.getNext(); 49. } 50. // 链表长度加一 51. length++; 52. } 53. }</pre>
--	--

remove()方法的三种实现：

顺序数组	<pre> 1. public void remove() { 2. // 如果列表为空 3. if(isEmpty()){ 4. System.out.println("List is empty"); 5. return; 6. } 7. // 将游标位置之后的元素依次向前移动一个位置 8. for (int i = cursor; i < length - 1; i++) { 9. array[i] = array[i + 1]; 10. } 11. // 列表长度减1 12. length--; 13. // 如果列表长度为0 14. if(length==0){ 15. // 游标位置设为-1 16. cursor=-1; 17. }else if(cursor==length){ 18. // 游标位置设为0 19. cursor=0; 20. } 21. }</pre>
单向链表	<pre> 1. public void remove() throws ListException { 2. // 检查列表是否为空 3. if(isEmpty()) { 4. // 抛出异常 if list is empty 5. throw new ListException("List is empty"); 6. } 7. Node<Character> temp=head; 8. while(true){ 9. // 找到当前游标位置的前一个节点 10. if(temp.getNext()==cursor){ 11. // 退出循环 12. break; 13. }else{ 14. temp=temp.getNext(); 15. } 16. }</pre>

	<pre> 17. // 将当前游标位置的节点从链表中移除 18. temp.setNext(cursor.getNext()); 19. 20. // 检查链表是否为空 21. if(isEmpty()){ 22. // 清空链表 23. clear(); 24. return; 25. // 如果链表为空, 则 cursor 变为头节点 26. }else if(cursor.getNext()==null){ 27. tail=temp; 28. cursor=head.getNext(); 29. } 30. // 将 cursor 指向新的节点 31. else cursor=temp.next; 32. 33. // 更新链表的长度 34. length--; 35. }</pre>
双向链表	<pre> 1. public void remove() throws ListException { 2. // 如果列表为空, 抛出异常 3. if(isEmpty()) { 4. throw new ListException("List is empty"); 5. } 6. // 如果当前节点是尾节点且前一个节点是头节点 7. if(cursor==tail&&cursor.getPrev()==head){ 8. clear(); 9. return; 10. } 11. // 如果当前节点是尾节点 12. if(cursor==tail) { 13. // 将前一个节点的下一个节点设为 null 14. cursor.getPrev().setNext(null); 15. // 更新尾节点为前一个节点 16. tail=cursor.getPrev(); 17. // 更新当前节点为头节点的下一个节点 18. cursor=head.getNext(); 19. } 20. // 如果当前节点是头节点的下一个节点 21. else if(cursor==head.getNext()) { 22. // 将头节点的下一个节点设为当前节点的下一个节点 23. head.setNext(cursor.getNext()); 24. // 更新当前节点为头节点的下一个节点 25. cursor=head.getNext(); 26. } 27. // 其他情况 28. else{ 29. // 将当前节点的前一个节点的下一个节点设为当前节点 // 的下一个节点 30. cursor.getPrev().setNext(cursor.getNext()); 31. // 将当前节点的下一个节点的前一个节点设为当前节点</pre>

	点的前一个节点
32.	cursor.getNext().setPrev(cursor.getPrev());
33.	// 更新当前节点为当前节点的下一个节点
34.	cursor=cursor.getNext();
35.	}
36.	// 更新列表长度
37.	length--;
38.	}

replace () 方法的三种实现：

顺序数组	<pre> 1. public void replace(Character newElement) throws ListException { 2. if(isEmpty()){ 3. insert(newElement); 4. return; 5. } 6. array[cursor] = newElement; 7. } </pre>
单向链表	<pre> 1. public void replace(Character newElement) throws ListException { 2. Node<Character> newElementNode=new Node<Character>(newElement); 3. Node<Character> temp=head; 4. 5. // 如果要替换的节点是头节点 6. if(cursor==head){ 7. // 在链表尾部插入新节点 8. insert(newElement); 9. return; 10. } 11. while(true){ 12. // 找到当前游标位置的前一个节点 13. if(temp.getNext()==cursor){ 14. // 退出循环 15. break; 16. }else{ 17. temp=temp.getNext(); 18. } 19. } 20. // 将当前游标位置的节点从链表移除，并将新节点与下一个节点相连 21. newElementNode.next=cursor.getNext(); 22. 23. // 检查链表是否为空 24. if(isEmpty()){ 25. // 更新尾节点为新节点 26. tail=newElementNode; 27. } 28. 29. // 设置当前节点为新节点 </pre>

	<pre> 30. cursor=newElementNode; 31. // 将前一个节点与新节点相连 32. temp.setNext(newElementNode); 33.}</pre>
双向链表	<pre> 1. public void replace(Character newElement) throws ListException { 2. // 创建新节点 3. Node<Character> newElementNode=new Node<Character>(newElement);; 4. // 如果当前节点是头节点或列表为空 5. if(cursor==head isEmpty()){ 6. insert(newElement); 7. return; 8. } 9. // 如果当前节点是尾节点 10. if(cursor==tail&&cursor.getPrev()==head){ 11. // 将新节点设为当前节点的下一个节点 12. newElementNode.setNext(cursor.getNext()); 13. // 将当前节点的下一个节点设为新节点 14. cursor.getNext().setPrev(newElementNode); 15. // 更新头节点的下一个节点为新节点 16. head.setNext(newElementNode); 17. } 18. // 如果当前节点是尾节点 19. else if(cursor==tail){ 20. // 将当前节点的前一个节点的下一个节点设为新节点 21. cursor.getPrev().setNext(newElementNode); 22. // 将新节点的前一个节点设为当前节点的前一个节点 23. newElementNode.setPrev(cursor.getPrev()); 24. // 更新尾节点为新节点 25. tail=newElementNode; 26. } 27. // 如果当前节点是头节点的下一个节点 28. else if(cursor==head.getNext()){ 29. // 将新节点设为当前节点的下一个节点 30. newElementNode.setNext(cursor.getNext()); 31. // 将当前节点的下一个节点设为新节点 32. cursor.getNext().setPrev(newElementNode); 33. // 更新头节点的下一个节点为新节点 34. head.setNext(newElementNode); 35. } 36. // 其他情况 37. else{ 38. // 将新节点设为当前节点的下一个节点 39. newElementNode.setNext(cursor.getNext()); 40. // 将当前节点的下一个节点设为新节点 41. cursor.getNext().setPrev(newElementNode); 42. // 将新节点的前一个节点设为当前节点的前一个节点 43. newElementNode.setPrev(cursor.getPrev()); 44. // 将当前节点的前一个节点的下一个节点设为新节点 45. cursor.getPrev().setNext(newElementNode);</pre>

	46.	// 更新头节点的下一个节点为新节点
	47.	head.setNext(newElementNode);
	48.	}
	49.	// 更新当前节点为新节点
	50.	cursor=newElementNode;
	51.	}

主函数:

```

1.  public static void main(String[] args) throws ListExceptio
    n{
2.      List<Character> list = new ArrayListDemo(512);
3.      //定义List 对象指向不同的类
4.      PrintWriter pw = new PrintWriter(System.out);
5.      int N=0;
6.      try (BufferedReader br = new BufferedReader(new Fil
    eReader("D:reportsource/list_testcase.txt"))) {
7.          String line;
8.          while ((line = br.readLine()) != null) {
9.              // 在这里处理每一行的文本
10.             execute(line, list);
11.             list.showStructure(pw);
12.         }
13.     } catch (IOException e) {
14.         e.printStackTrace();
15.     }
16.     pw.close();
17. }
18. public static void execute(String example, List<Charact
    er> list) throws ListException {
19.
20.     int i=0;
21.     while(i<example.length()){
22.         char a = example.charAt(i);
23.         switch (a) {
24.             case '+':
25.                 char insertchar = example.charAt(++i);
26.                 //插入元素, 需要多读入一位字符, 下面
                replace 同理
27.                 try{list.insert(insertchar);
28.                 }catch (ListException e){
29.                     System.out.println("List is full");
30.                 }
31.                 break;
32.             case '-':
33.                 list.remove();

```

```

34.             break;
35.             case '=':
36.                 char replacetchar = example.charAt(++i);
37.                 list.replace(replacetchar);
38.                 break;
39.             case '#':
40.                 list.gotoBeginning();
41.                 break;
42.             case '*':
43.                 list.gotoEnd();
44.                 break;
45.             case '>':
46.                 list.gotoNext();
47.                 break;
48.             case '<':
49.                 list.gotoPrev();
50.                 break;
51.             case '~':
52.                 list.clear();
53.                 break;
54.         }
55.
56.         i+=2;
57.     }
58. }

```

4.运行结果：

对三种存储方式实现程序的测试都得到了以下相同的结果：

```

a {capacity=512,length=1,cursor=0}
0 v 0 N p G {capacity=512,length=6,cursor=0}

进程已结束，退出代码为 0

```

(给出清晰的最后两行结果便于初步检验，完整结果见下)

(完整运行结果, 经比对与所给结果完全一致)

5.总结收获：

下面是关于三种实现方式不同方法时间复杂度的总结：

	insert	remove	replace	clear	isEmpty	isFull	gotoBegining
顺序存储	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
单向链表	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
双向链表	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
	gotoEnd	gotoPrev	gotoNext	getCursor	showStructure	moveToNth	find
顺序存储	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
单向链表	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
双向链表	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$

收获：

- 1) 能够自行定义数据结构，利用不同存储方式实现要求 ADT
- 2) 更加了解顺序数组、单向链表、双向链表各不同存储方式在时间与空间复杂度的优劣比较
- 3) 解决问题过程中有许多对边界值的判断，一定程度上提高了处理边界的能力

任务二

1.题目：创建一个可自动调整空间大小的 List 数据结构

观察任务 1 中基于数组实现的线性表的测试用例的运行结果，发现大部分时候空间的使用率是不高的（length 和 capacity 的比值反映了这一事实），而且还存在有空间不够用

的例外发生。当然，基于链式存储实现的线性表则不存在此类问题。为了解决空间利用率以及空间不够用的问题，任务 2 将使用动态调整的方式改善数组空间的大小，方案可以有很多种，但在本次实验中将采用如下的调整方案，具体步骤如下：

- ① 使用 capacity 表示当前线性表的最大容量（即最多能够存储的线性表元素个数）；
- ② 初始情况下，capacity=1；
- ③ 当插入元素时线性表满，那么就重新生成一个容量为 $2 \times \text{capacity}$ 的数组，将原数组中的 capacity 个元素拷贝到新数组中，让新数组成为当前线性表的存储表示；
- ④ 当删除元素之后，如果当前线性表中的元素个数 length 是 capacity 的四分之一时，则重新生成一个容量为 $\text{capacity}/2$ 的数组，将原数组中的 length 个元素拷贝到新数组中，让新数组成为当前线性表的存储表示。

2.数据与算法设计：

ResizingAlist 的实现：

根据题意该 ResizingAlist 类与任务一的 ArrayListDemo 基本一致，只是需要修改 insert、remove 方法的实现，此外在初始化时要让 capacity=1；

- 1) 为了简化改变长度的操作，定义 resize 方法，传入新的长度 newlength，将原数组拷贝进新建 newlength 长度的数组中。
- 2) 在 insert 方法中，先判断数组是否为满，满则定义 $\text{newlength} = 2 \times \text{capacity}$ ，并调用 resize 方法，在进行后续插入操作
- 3) 在 remove 方法中，先进行之前的删除操作，若删除完成后数组长度 = capacity 的 $1/4$ ，则定义 $\text{newlength} = \text{capacity}/2$ ，并调用 resize 方法

生成对比图表：

- 1) 为获取 list 的空间利用率，在两个数据结构中都定义 getSpaceOccupancy 方法，传回 $\text{length}/\text{capacity}$ 的浮点数值。
- 2) 利用第一次实验中学习的 JFreeChart 接口绘图，X 轴为已经处理的行数，Y 轴为空间利用率

3.部分代码说明：

1) resize:

```
1. private void resize(int newlength){
2.     //新建长度数组
3.     Character[] newarray = new Character[newlength];
4.     //将原数组元素拷贝进新数组
5.     for(int i=0;i<length;i++){
6.         newarray[i] = array[i];
7.     }
8.     //将新数组赋给原数组
9.     array = newarray;
10.    //更新容量
11.    capacity = newlength;
```



```
12.    }
13.
```

2) insert:

```
1.    public void insert(Character newElement) throws ListExc
      eption {
2.        if(isFull()){
3.            // 数组满则扩容
4.            resize(capacity*2);
5.        }
6.        for(int i=length;i>cursor+1;i--){
7.            array[i] = array[i-1];
8.        }
9.        array[++cursor] = newElement;
10.       length++;
11.    }
```

3) remove:

```
1.    public void remove() {
2.        if(isEmpty()||cursor==-1){
3.            System.out.println("List is empty");
4.            return;
5.        }
6.        for (int i = cursor; i < length - 1; i++) {
7.            array[i] = array[i + 1];
8.        }
9.        array[length-1] = null;
10.       length--;
11.       if(length==0){
12.           cursor=-1;
13.       }else if(cursor==length){
14.           cursor=0;
15.       }
16.       // 缩容
17.       if(length>0&&length==capacity/4){
18.           resize(capacity/2);
19.       }
20.    }
```


4.运行结果展示：

```
V J U t v 0 d a w c i b l {capacity=16,length=13,cursor=1}
b U n w I y l t B h e a t e g m C g v j {capacity=32,length=20,cursor=2}
a l o b m a z N J b b j s U n w I y l t B h e a t e g l w w {capacity=32,length=30,cursor=0}
n g h K b z s r k i X W p s l r y h o b m a z N J b b j s U n w I y l t B h e a t e g l w j g {capacity=64,length=47,cursor=0}
u d h S o y e a s c b z x j R B w q {capacity=32,length=18,cursor=17}
x v h S o y e a s c b z x j R B w P t {capacity=32,length=19,cursor=18}
G n f c P r s {capacity=16,length=8,cursor=1}
Z F n o k B y z w D l p h b f e c P r s n y g t t h q V p {capacity=32,length=29,cursor=28}
I D o S o n t w p n n y e o v b k B y z w D l p h b f e c P r s n y g t t h V C l t c a g v m {capacity=64,length=47,cursor=45}
T n p g A Q m n C l z m h x {capacity=16,length=14,cursor=13}
R r c J q p u V i j j t Q y e c M f u t w n p g A Q m n C l z m h x e r v q H r j {capacity=64,length=41,cursor=7}
s d v y e a o u x j {capacity=16,length=10,cursor=9}
s y j o c f c b x l b w l h b d v y e a o u x j K T w u n p l d {capacity=32,length=32,cursor=31}
o e p H k Z m c o n k o z o x y j X t o c f c b x l b w l h b d v y e a o u x j K T w u n p l B G g w e {capacity=64,length=52,cursor=2}
A o k b j n a f y j z T C {capacity=16,length=13,cursor=2}

s d v y e a o u x j {capacity=16,length=10,cursor=9}
s y j o c f c b x l b w l h b d v y e a o u x j K T w u n p l d {capacity=32,length=32,cursor=31}
o e p H k Z m c o n k o z o x y j X t o c f c b x l b w l h b d v y e a o u x j K T w u n p l B G g w e {capacity=64,length=52,cursor=2}
A o k b j n a f y j z T C {capacity=16,length=13,cursor=2}
E h s r q t e j h f r X q k m I g o z a k b l q j n a f y j z T C m b z z e g {capacity=64,length=39,cursor=0}
p y r q n s Y N q t e j h f r X q k m I g o z a k b l q j n a f y j z T C m b z z e g u o u w j c q u {capacity=64,length=51,cursor=50}
P Y w e t a t f d r j r a t y r q n s Y N q t e j h f r X q k m I g o z a k b l q j n a f y j z T C m b z z e g u o u w j T z y o u z n h {capacity=128,length=70,cursor=0}
Q b {capacity=2,length=2,cursor=1}
c m {capacity=4,length=2,cursor=1}
q j l j c J c k f H j o x i t h n L H t m {capacity=32,length=21,cursor=18}
d H D V R d e w v d r g g K d j c J c k f H j o x i t h n L H t m t a v n g v m w k p U T {capacity=64,length=45,cursor=5}
I T j g f n o t e k u v Y D t V R d e w v d r g g K d j c J c k f H j o x i t h n L H t m t a v n g v m w k p U T j u r q I k F g {capacity=128,length=66,cursor=1}
Empty List {capacity=1,length=0,cursor=-1}
A r n u x a m g v u P e h r Z Q t x v q g {capacity=32,length=21,cursor=0}
o x y Y d r k d o i z r n u x a m g v u P e h r Z Q t x v q P i v e n s j e c h t c {capacity=64,length=42,cursor=41}
z f G m z f l y Y d r k d o i z r n u x a m g v u P e h r Z Q t x v q P i v e n s j e c h t M O e C m c s r o n g h k r p n y g r f m V {capacity=128,length=68,cursor=3}
r c m {capacity=4,length=3,cursor=2}
W J L c m B z c q j q q L Z {capacity=32,length=14,cursor=0}
z l e d g p C b C J c u v X m B z c q j J C H {capacity=32,length=23,cursor=0}
G x w j l z v e c k r e l e d g p C b C J c u v X m B z c q j J M z J w t p K {capacity=64,length=40,cursor=38}
l x g w n r e n c a u x p O x v m j l z v e c k r e l e d g p C b C J c u v X m B z c q j J H z J w t p K a M v J o n S z t e {capacity=64,length=63,cursor=5}
Empty List {capacity=1,length=0,cursor=-1}
w i E h w r O m F g r s x s n z V r n {capacity=32,length=19,cursor=6}
z a a h y r n S K L E h w u r d r O H m F g r s x s n z V r n q z l l n {capacity=64,length=37,cursor=8}
I f f y v f h t a P v l a h y r n S K f y b d L E h w u r d r O H m F g r s x s n z V r n q z l l n v d Y y j g W q {capacity=64,length=59,cursor=1}
f f y r r E g j z f y t T W R x e m {capacity=32,length=18,cursor=16}
l q s m i h t V d h k c d p f s m c k t f v b e s y r r E g j z f y t T W R x n m N {capacity=64,length=43,cursor=42}
a k n n j J i d s l l u n b s m g q P y p d z {capacity=32,length=24,cursor=23}
F t d g p x v M a j P j o v w {capacity=16,length=15,cursor=2}
l e V K e w k y c y P t d g p x v M a j P j o v w l u r H {capacity=32,length=30,cursor=29}

a k n n j J i d s l l u n b s m g q P y p d z {capacity=32,length=24,cursor=23}
F t d g p x v M a j P j o v w {capacity=16,length=15,cursor=2}
L e j V k e w k y c y P t d g p x v M a j P j o v l u r H {capacity=32,length=30,cursor=29}
K o o q U w h f f e g j V k e w k y c y P t d g p x v M a j P j o v l u r V q j g q q j d f a c {capacity=64,length=49,cursor=48}
T I K t y z f o q U w h f f e g j V k e w k y c y P t d g p x v M a j P j o v l u r V q j g q q j d f a R M U C c H m l {capacity=64,length=61,cursor=0}
A w D a F i j h b a c d v f l g c e s a a k y I K t y z f o q U w h f f e g j V k e w k y c y P t d g p x v M a j P j o v l u r V q j g q q j d f a R M U C c H m l {capacity=128,length=74,cursor=0}
G a u a v q P R K x s v m O B v i j h b a c d v f l g c e s a a k y I K t y z f o q U w h f f e g j V k e w k y c y P t d g p x v M a j P j o v l u r V q j g q q j d f a R M U C c H m l {capacity=128,length=74,cursor=0}
f G d A o d c {capacity=8,length=5,cursor=2}
r E t x e {capacity=8,length=5,cursor=2}
G l b p v Z z g g c E t m c t d l x e j t e i n R g R {capacity=32,length=27,cursor=0}
l o y t z S m f d I J v Z z g g c E t m c t d l x e j t e i n R g R c q n e z n u w q s {capacity=64,length=44,cursor=2}
O D v n j M i C W s v c u k o y t v z S m f d I J v Z z g g c E t m c t d l x e j t e i n R g R c q n e z n u w S V m a c {capacity=64,length=62,cursor=5}
C M c k Q v q C f v G u k C H C s O v n j i j v C W s v c u k o y t v z S m f d I J v Z z g g c E t m c t d l x e j t e i n R g R c q n e z n u w S V m a c c c K i {capacity=128,length=74,cursor=0}
P t B v C i r {capacity=8,length=7,cursor=6}
H s m j d G d h Y g r y v k t B v C i R m v Z i m m y s y {capacity=32,length=29,cursor=4}
c h d y g a m u u a c t f y j s m j d G d h Y g r y v k t B v C i R m v Z i m m y s a u k q {capacity=64,length=48,cursor=1}
y d y a I p j U z d y y g a m u u a c t f y j s m j d G d h Y g r y v k t B v C i R m v Z i m m y s a u k q x f x j y z r o v i b {capacity=128,length=67,cursor=1}
S h v a q I w z i c M i p p a g i I p j U z d y y g a m u u a c t f y j s m j d G d h Y g r y v k t B v C i R m v Z i m m y s a u k q x f x j y z r o t v j b K {capacity=128,length=67,cursor=1}
e z y g z d v j x h v a q I w z i c M i p p a g i I p j U z d y y g a m u u a c t f y j s m j d G d h Y g r y v k t B v C i R m v Z i m m y s a u k q x f x j y z r o t v j V {capacity=128,length=67,cursor=1}
v j b l y a a s b M Z x z Y o C t d w g z d v j x h v a q I w z i c M i p p a g i I p j U z d y y g a m u u a c t f y j s m j d G d h Y g r y v k t B v C i R m v Z i m m y s a u k q x f x j y z r o t v j V {capacity=128,length=67,cursor=1}
O c c u O i H l b G {capacity=16,length=10,cursor=4}
e p l c p u p b m K c i u w H l b G m r d i L x y x x f {capacity=32,length=28,cursor=27}
j a a l x l l c c p u p b m K c i u w H l b G m r d i L x y x x f f l c V d f r i i {capacity=64,length=42,cursor=41}
O z r u y O C a C o c a l x l l c c p u p b m K c i u w H l b G m r d i L x y x x f f l c V d f r i i x l u b y j t g {capacity=64,length=59,cursor=58}
v z T q h l s p d q r k u y O C a C o c a l x l l c c p u p b m K c i u w H l b G m r d i L x y x x f f l c V d f r i i x l u b y j Q R o r f j d {capacity=128,length=74,cursor=0}
h u y C t T a Z r O n m h p z T q h l s p d q r k u y O C a C o c a l x l l c c p u p b m K c i u w H l b G m r d i L x y x x f f l c V d f r i i x l u b y j Q R o r f j d {capacity=128,length=74,cursor=0}
l e o B g g a U b i a B Z F w e o U u c Q e a R C t T a Z r O n m h p z T q h l s p d q r k u y O C a C o c a l x l l c c p u p b m K c i u w H l b G m r d i L x y x x f f l c V d f r i i x l u b y j Q R o r f j d {capacity=128,length=74,cursor=0}
x e w d J E c h x b s l f e Y M B T g g a U b i a B Z F w e o U u c Q e a R C t T a Z r O n m h p z T q h l s p d q r k u y O C a C o c a l x l l c c p u p b m K c i u w H l b G m r d i L x y x x f f l c V d f r i i x l u b y j Q R o r f j d {capacity=128,length=74,cursor=0}
K k m q q j w o p j N q v h e e w d l J E c h x b s l f e Y M B T g g a U b i a B Z F w e o U u c Q e a R C t T a Z r O n m h p z T q h l s p d q r k u y O C a C o c a l x l l c c p u p b m K c i u w H l b G m r d i L x y x x f f l c V d f r i i x l u b y j Q R o r f j d {capacity=128,length=74,cursor=0}
x d i W {capacity=64,length=4,cursor=3}
```

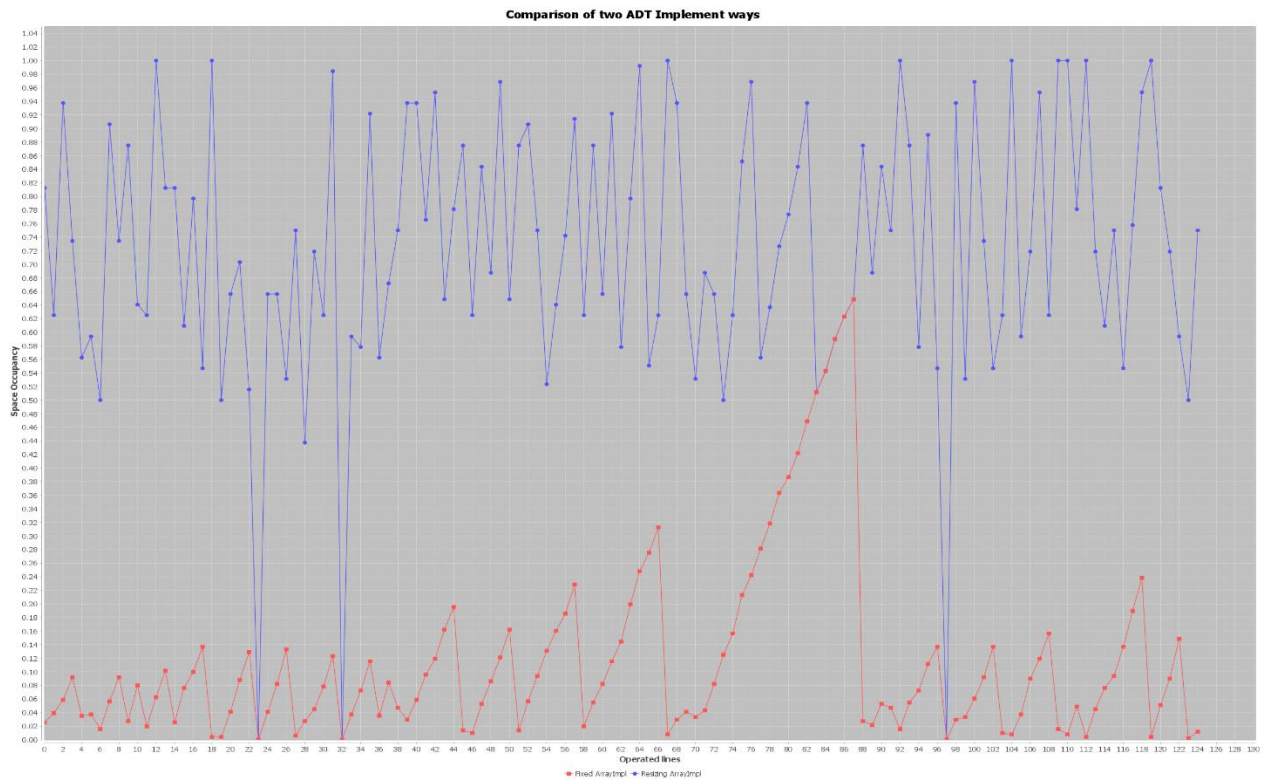
(由于数据规模和图片像素问题中间略，给出最后几行的清晰结果)

```
i s {capacity=2,length=2,cursor=1}
P p e h S y l S f p z k n l q p e z c E Q e z G t w {capacity=32,length=26,cursor=4}
W r k c g w h l h n v t p e h S g y l S f p z k n l q p e z c E Q e z G t e n j x j e l g r {capacity=64,length=46,cursor=44}
J v b A q o w F x a v p a z g x r V v t b c n i r k c g w h l h n v t p e h S g y l S f p z k n l q p e z c E Q e z G t e n j x j e l a {capacity=2,length=1,cursor=0}
O v O n p G {capacity=8,length=6,cursor=0}

进程已结束，退出代码为 0
```

下面是空间利用率的对比图：

(红色为固定长度数组，蓝色为动态数组)



5.总结与收获：

使用该动态调整数组大小方法能够有效提高数组存储方式的空间利用率，学习并成功应用了该方法。

任务三

1.题目：栈

众所周知，栈虽然是一个操作受限的线性表，但是其用途却很广泛，栈的数据结构实现非常简单，因此我们只从应用层面熟悉栈。请完成下面三个子任务。

①递归是一种解决很多复杂问题最简单的思想方法，而任何编程语言对递归程序的支持都是通过栈实现的。请利用课堂上讲解的“Hanoi 塔”问题的非递归转化方法完成对递归快速排序的非递归转化。

②算术混合运算表达式的计算。表达式不仅能处理整数，还需要处理小数。表达式中涉及的运算符包括+、-、*、/、^(指数)。表达式可以包含括号（只包含圆括号）嵌套，因此要处理括号匹配失败的情形。

③当我们在使用很多软件时都有类似“undo”功能，比如 Web 浏览器的回退功能、文本编辑器的撤销编辑功能。这些功能都可以使用 Stack 简单实现，但是在现实中浏览器的回退功能也好，编辑器的撤销功能也好，都有一定的数量限制。因此我们需要的不是一个普通的 Stack 数据结构，而是一个空间有限制的 Stack，虽然空间有限，但这样的 Stack 在入栈时从不会溢出，因为它会采用将最久远的记录丢掉的方式让新元素入栈，也就是说总是按照规定的数量要求保持最近的历史操作。比如栈的空间是 5，当 a\b\c\d\e 入栈之后，如果继续让元素 f 入栈，那么栈中的元素将是 b\c\d\e\f。请设计一个满足上面要求的 LeakyStack 数据结构，要求该数据结构的每一个操作的时间复杂度在最坏情形下都必须满足 $O(1)$ 。

2.快速排序：

2.1 数据与算法设计

与报告一中未优化版快速排序的实现逻辑类似，但是将递归转为栈，转换部分操作如下：

- (1) 初始化栈,入栈整个数组的左右索引
- (2) 当栈不空时，弹出一个区间的左右索引
- (3) 如果区间长度大于 1,选择中间位置元素作为 pivot,进行 partition；不够大则直接 continue
- (4) 根据 partition 结果,将左右区间重新入栈用于后续排序
- (5) 重复步骤 2-4 直到栈空

2.2 部分代码说明

```
1. package task3;
2.
3. import java.util.Stack;
4.
5. public class QuickSort2 {
6.
7.     // 使用堆栈实现快速排序
8.     static void quickSort(int[] arr) {
9.         Stack<Integer> stack = new Stack<>();
10.    //入栈整个数组的左右索引
11.        stack.push(0);
12.        stack.push(arr.length);
13.    //栈不空则持续操作
14.        while (!stack.isEmpty()) {
15.            int end = stack.pop();
16.            int start = stack.pop();
17.            if (end - start < 2) {
18.                continue;
19.            }
```

```

20.
21.         int p = start + ((end - start) / 2);
22.         p = partition(arr, p, start, end);
23.
24.         stack.push(p + 1);
25.         stack.push(end);
26.
27.         stack.push(start);
28.         stack.push(p);
29.     }
30. }
31.
32.     // 分组后, 返回新的分界点位置
33.     private static int partition(int[] arr, int position, i
        nt start, int end) {
34.         int l = start;
35.         int h = end - 2;
36.         int piv = arr[position];
37.         swap(arr, position, end - 1);
38.
39.         while (l < h) {
40.             if (arr[l] < piv) {
41.                 l++;
42.             } else if (arr[h] >= piv) {
43.                 h--;
44.             } else {
45.                 swap(arr, l, h);
46.             }
47.         }
48.         int idx = h;
49.         if (arr[h] < piv) {
50.             idx++;
51.         }
52.         swap(arr, end - 1, idx);
53.         return idx;
54.     }
55.
56.     // 交换 arr 中两个元素
57.     private static void swap(int[] arr, int i, int j) {
58.         int temp = arr[i];
59.         arr[i] = arr[j];
60.         arr[j] = temp;
61.

```

2.3 运行结果展示：

```

QuickSortTest
E:\java帮助文档\bin\java.exe "-javaagent:E:\java环境\IntelliJ IDEA Community E
请输入排序数组大小:
20
请输入排序数组:
0 2 3 4 5 5 5 2 29 64 56 32 33 0 2 4 1 12 58 55
排序结果为:
0 1 2 2 2 3 4 5 5 6 9 12 29 32 33 55 55 56 58 64
进程已结束，退出代码为 0

```

3.计算器：

3.1 数据与算法设计

主程序逻辑：

- (1) 首先利用栈判断传入的表达式左右括号是否匹配，如果不匹配则抛出自定义 UnEqual 异常
- (2) 初始化两个栈，一个 numStack 存放数字，一个 opStack 存放运算符
- (3) 遍历表达式字符
 - 遇到数字字符入数栈
 - 遇到左括号入运算符栈
 - 遇到右括号开始运算，直至遇到符号栈顶为左括号，最后将其弹出
 - 遇到运算符运算符与栈顶运算符比较优先级，如果当前栈顶运算符优先级高则运算，否则将新运算符入栈
- (4) 遍历结束后,运算栈中剩余的元素

在具体实现中，还需注意：

- 1) 由于表达式中数字包含小数，因此在将数字入栈时，必须将完整的小数截取，这里使用 StringBuilder，一旦遇到数字，就定义一个 StringBuilder 对象，并将后面的小数点与数字均拼接上去，直至遇到运算符或到达表达式结尾
- 2) 在运算方法中，因为运算逻辑，要注意先弹出的为减数/除数，不能混乱运算顺序

3.2 部分代码说明：

```

1.
   // 计算给定表达式的值
2. public static double calculate(String expression) throws Un
   equalException {
3.
4.   // 检查表达式的括号是否匹配
5.   if (isMatch(expression)) {
6.

```



```

7.      // 数字栈,用于存储操作数
8.      Stack<Double> numStack = new Stack<>();
9.      // 运算符栈,用于存储运算符
10.     Stack<Character> opStack = new Stack<>();
11.
12.     // 遍历表达式的每个字符
13.     for (int i = 0; i < expression.length(); i++) {
14.         char c = expression.charAt(i);
15.
16.         // 如果当前字符是数字,提取出整个数字加入数栈
17.         if (Character.isDigit(c) || c == '.') {
18.             StringBuilder sb = new StringBuilder();
19.             while (i < expression.length() && (Character.isDigit(
                expression.charAt(i)) || expression.charAt(i) == '.')) {
20.                 sb.append(expression.charAt(i++));
21.             }
22.             i--;
23.             numStack.push(Double.parseDouble(sb.toString()));
24.         }
25.         // 当前字符是左括号,入运算符栈
26.         else if (c == '(') {
27.             opStack.push(c);
28.
29.         // 当前字符是右括号,计算栈顶运算符
30.         } else if (c == ')') {
31.             while (opStack.peek() != '(') {
32.                 performOperation(numStack, opStack);
33.             }
34.             opStack.pop();
35.
36.         // 当前字符是运算符
37.         } else if (c == '+' || c == '-'
            || c == '*' || c == '/' || c == '^') {
38.             // 当栈不空,且栈顶运算符优先级大于等于当前运算符,计算栈
                顶运算符
39.             while (!opStack.isEmpty() && precedence(opStack.pee
                k()) >= precedence(c)) {
40.                 performOperation(numStack, opStack);
41.             }
42.             // 当前运算符入栈
43.             opStack.push(c);
44.         }
45.     }
46.
47.     // 遍历结束,计算栈中剩余的运算符

```

```

48.     while (!opStack.isEmpty()) {
49.         performOperation(numStack, opStack);
50.     }
51.
52.     // 返回表达式计算结果
53.     return numStack.pop();
54.
55. } else {
56.     // 括号不匹配, 抛出异常
57.     throw new UnequalException("括号不匹配");
58. }

1. private static void performOperation(Stack<Double> numStack, Stack<Character> opStack) {
2.
3.     // 从运算符栈顶弹出当前运算符
4.     char op = opStack.pop();
5.
6.     // 从数栈弹出运算的两个数
7.     double num2 = numStack.pop();
8.     double num1 = numStack.pop();
9.
10.    double result = 0;
11.
12.    // 根据不同的运算符, 进行相应计算
13.    switch(op) {
14.        case '+':
15.            result = num1 + num2;
16.            break;
17.        case '-':
18.            result = num1 - num2;
19.            break;
20.        // ... 省略其他运算符情况...
21.
22.        // 计算结果入数栈
23.        numStack.push(result);
24.    }
25.}

```

3.3 运 行 结 果 展 示

```
运行: CalculatorTest x
E:\java帮助文档\bin\java.exe "-javaagent:E:\java环境\IntelliJ IDEA Community Ed
请输入表达式:3*5.1-2.3*(2.3-0.3)
The number of ( and ) in this expression are not equal!
进程已结束，退出代码为 0
```

```
CalculatorTest x
E:\java帮助文档\bin\java.exe "-javaagent:E:\java环境\Int
请输入表达式:3*5.1-2.3*(2.3-0.3)
计算结果为: 10.7
进程已结束，退出代码为 0
```

4. LeakyStack:

4.1 数据与算法设计

注意到题目要求各方法的最坏情况下时间复杂度为 $O(1)$ ，因此考虑在该任务中，采用循环数组来实现该 LeakyStack，设计数据除数组外，定义 `top` 指向当前栈顶的下一个元素，定义 `bottom` 为当前栈中最先入栈的元素。这样就能做到入栈与出栈、判断栈空、栈满都只是通过简单的 `top` 与 `bottom` 运算得出，时间复杂度保持 $O(1)$ 。

而要满足题目中要求的 `push` 操作，只需要在栈满时，除了基本的入栈操作外，更新 `bottom` 的值即可。

4.2 部分代码说明:

```
1. public void push(T item) {
2.     if (size == capacity) {
3.         // 栈已满，需要"泄漏"最旧的元素
4.         bottom = (bottom + 1) % capacity;
5.     } else {
6.         size++;
7.     }
8.     //将item入栈
9.     stack[top] = item;
10.    //注意循环数组中bottom和top的更新方式
11.    top = (top + 1) % capacity;
12. }
13.
14. public T pop() {
```

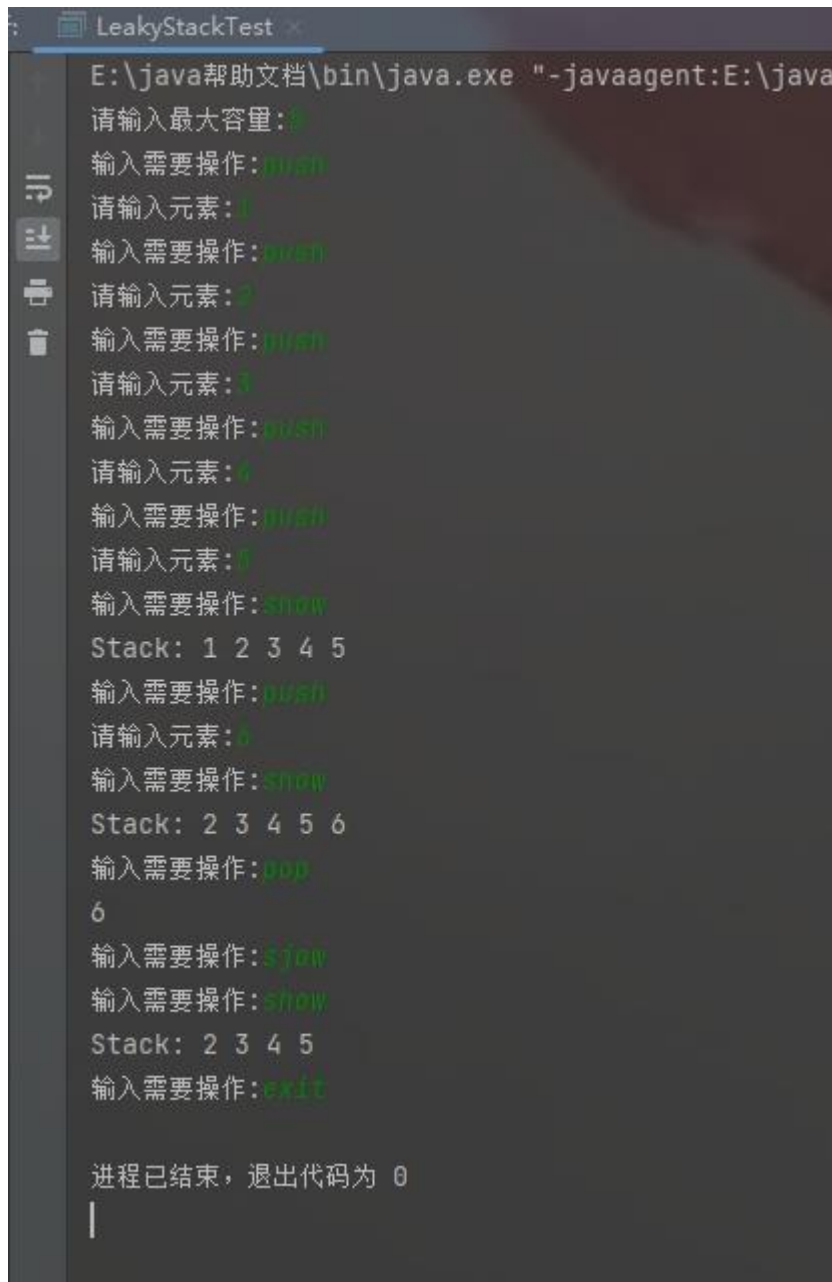


```

15.         if (isEmpty()) {
16.             return null;
17.         }
18.         //注意循环数组中bottom 和top 的更新方式
19.         top = (top - 1 + capacity) % capacity;
20.         size--;
21.         //将栈顶元素出栈
22.         return stack[top];
23.     }

```

4.3 运行结果展示：



```

LeakyStackTest x
E:\java帮助文档\bin\java.exe "-javaagent:E:\java
请输入最大容量:5
输入需要操作:push
请输入元素:1
输入需要操作:push
请输入元素:2
输入需要操作:push
请输入元素:3
输入需要操作:push
请输入元素:4
输入需要操作:push
请输入元素:5
输入需要操作:show
Stack: 1 2 3 4 5
输入需要操作:push
请输入元素:6
输入需要操作:show
Stack: 2 3 4 5 6
输入需要操作:pop
6
输入需要操作:show
Stack: 2 3 4 5
输入需要操作:exit

进程已结束，退出代码为 0
|

```

5.总结与收获：

- 1) 学会了使用栈构建混合表达式计算器
- 2) 学习了用栈代替递归的思想
- 3) 更加深刻地掌握栈先入后出的特性

任务四

1.题目：基数排序

使用自定义的队列数据结构实现对某一个数据序列的排序(采用基数排序)，其中对待排序数据有如下的要求: ①当数据序列是整数类型的数据的时候，数据序列中每个数据的位数不要求等宽，比如: 1、21、12、322、44、123、2312、765、56 ②当数据序列是字符串类型的数据的时候，数据序列中每个字符串都是等宽的，比如: "abc","bde","fad","abd","bef","fdd","abe"

2.数据与算法设计：

1) 解决该任务要了解基数排序的逻辑，首先要根据比较对象组成部分的特性建立相应数目个有序队列，按照优先级从低到高依次提取各比较对象的组成部分，并依照该组成部分将比较对象进入相应队列，待所有对象都入队后再全部出队，重复该过程直至将比较对象优先级最大的组成成分也处理完毕，此时出队后形成的序列即为有序序列。这是因为队列先入先出的特性，某种意义上可以保证序列在几次入队出队后的相对位置不变。

2) 按照该思想，当比较数字时，其组成成分的优先级即从个位数字开始到最高位逐渐升高。

3) 还需注意的是，由于数字位数不同，为了比较，需要首先将所有数字以高位置零的方式扩充到相同位数。

4) 比较字符串时，其组成成分从末尾字符开始到首位字符逐渐升高。

具体实现：

1) 以数组为存储方式自定义 Queue 类，可进行入队、出队操作。

2) 定义 RadixSort 类，其中包含对数字和对字符串的两个比较方法，在对数字的比较中，建立 10 个队列，对应各位上的数字；而在对字符串的比较方法中，建立 52 个队列，分别对 26 个大写与 26 个小写字母。接下来按照上面提到实现逻辑进行编写即可。

3) 在主测试函数中，读取文件的每一行，以空格为分割将数字/字符串存入数组中，再调用 RadixSort 进行排序操作，每处理一行就输出。

3.部分代码说明：

```
1. public class RadixSort {
2.     //使用基数排序算法对整数数组进行排序
3.     public static void radixSort(int[] arr) {
4.         // 找出最大数的位数
5.         int maxNumber = findMaxNumber(arr);
6.         int maxLength = Integer.toString(maxNumber).length(
            );
    }
```

```

7.
8.      // 对每一位进行排序
9.      for (int digit = 0; digit < maxLength; digit++) {
10.         // 使用队列数组作为桶
11.         Queue<Integer>[] buckets = new Queue[10];
12.         for (int i = 0; i < buckets.length; i++) {
13.             buckets[i] = new Queue<>();
14.         }
15.
16.         // 按当前位数分配到桶中
17.         for (int number : arr) {
18.             int bucketIndex = (number / (int) Math.pow(
19.                 10, digit)) % 10;
20.             buckets[bucketIndex].enqueue(number);
21.         }
22.         // 从桶中收集数字
23.         int arrayIndex = 0;
24.         for (Queue<Integer> bucket : buckets) {
25.             while (!bucket.isEmpty()) {
26.                 arr[arrayIndex++] = bucket.dequeue();
27.             }
28.         }
29.     }
30. }
31.
32. // 使用基数排序算法对字符串数组进行排序
33. public static void radixSort(String[] arr) {
34.     int maxLength = arr[0].length();
35.
36.     // 对每一位进行排序
37.     for (int digit = maxLength - 1; digit >= 0; digit--
38.         ) {
39.         Queue<String>[] buckets = new Queue[52]; // 大小
40.         // 写字母共 26 个
41.         for (int i = 0; i < buckets.length; i++) {
42.             buckets[i] = new Queue<>();
43.         }
44.         for (String s : arr) {
45.             // 将字符串 s 放入对应队列中
46.             int bucketIndex;
47.             if(s.charAt(digit) >= 'A' && s.charAt(digit)
48.                 ) <= 'Z'){
49.                 bucketIndex = s.charAt(digit) - 'A';

```

```

48.         }else{
49.             bucketIndex = s.charAt(digit) -
        'a' + 26;
50.         }
51.         buckets[bucketIndex].enqueue(s);
52.     }
53.
54.     int arrayIndex = 0;
55.     for (Queue<String> bucket : buckets) {
56.         while (!bucket.isEmpty()) {
57.             arr[arrayIndex++] = bucket.dequeue();
58.         }
59.     }
60. }
61. }
62.
63. //找出数组中的最大数
64. private static int findMaxNumber(int[] arr) {
65.     int max = arr[0];
66.     for (int i = 1; i < arr.length; i++) {
67.         if (arr[i] > max) {
68.             max = arr[i];
69.         }
70.     }
71.     return max;
72. }
73. }

```

主测试方法：

```

1. public static void main(String[] args) {
2.     try {
3.         // 创建一个File 对象，表示要读取的文件
4.         File file = new File("D:/reportsource/radixSort
        1.txt");
5.         // 创建一个Scanner 对象，用于读取文件内容
6.         Scanner scanner = new Scanner(file);
7.         // 创建一个File 对象，表示要输出的结果文件
8.         File outputFile = new File("D:/reportsource/rad
        ixSortresult1.txt");
9.         // 循环读取文件的每一行
10.        while (scanner.hasNextLine()) {
11.            // 读取并保存当前行的内容
12.            String line = scanner.nextLine();
13.            // 使用空格分割行中的数据，保存为字符串数组
14.            String[] dataArray = line.split(" ");

```

```

15.          // 创建一个整数数组，用于保存分割后的数据
16.          int[] numbers = new int[dataArray.length];
17.          // 将字符串数组中的数据转换为整数，保存到
           numbers 数组中
18.          for (int i = 0; i < dataArray.length; i++)
           {
19.              numbers[i] = Integer.parseInt(dataArray
           [i]);
20.          }
21.          // 调用 RadixSort 类的静态方法对 numbers 数组进
           行排序
22.          RadixSort.radixSort(numbers);
23.          try {
24.              // 创建一个 FileWriter 对象，用于向结果文件
           中写入数据
25.              FileWriter writer = new FileWriter(outp
           utFile, true);
26.              // 遍历排序后的 numbers 数组
27.              for (int i = 0; i < numbers.length; i++
           ) {
28.                  // 在控制台和结果文件中打印排序后的数据
29.                  System.out.print(numbers[i] + " ");
30.                  writer.write(String.valueOf(numbers
           [i]));
31.                  // 在结果文件中添加空格分隔符（除最后
           一个数据外）
32.                  if (i < numbers.length - 1) {
33.                      writer.write(" ");
34.                  }
35.              }
36.              // 在控制台和结果文件中换行
37.              System.out.println();
38.              writer.write("\n");
39.              writer.close();
40.          } catch (IOException e) {
41.              e.printStackTrace();
42.          }
43.      }
44.      scanner.close();
45.  } catch (FileNotFoundException e) {
46.      e.printStackTrace();
47.  }
48.  try {
49.      // 创建一个 BufferedReader 对象，用于读取文件内容

```

```

50.         BufferedReader reader = new BufferedReader(new
           FileReader("D:/reportsource/radixSort2.txt"));
51.         // 创建一个File 对象，表示要输出的结果文件
52.         File outputFile2 = new File("D:/reportsource/radixSortresult2.txt");
53.         // 读取并保存文件的每一行
54.         String line;
55.         while ((line=reader.readLine()) != null) {
56.             // 使用空格分割行中的数据，保存为字符串数组
57.             String[] dataArray2 = line.split(" ");
58.             // 创建一个字符串数组，用于保存分割后的数据
59.             String[] strings = dataArray2;
60.             // 调用RadixSort 类的静态方法对 strings 数组进行排序
61.             RadixSort.radixSort(strings);
62.             try {
63.                 // 创建一个FileWriter 对象，用于向结果文件中写入数据
64.                 FileWriter writer = new FileWriter(outputFile2, true);
65.                 // 遍历排序后的 strings 数组
66.                 for (int i = 0; i < strings.length; i++) {
67.                     // 在控制台和结果文件中打印排序后的数据
68.                     writer.write(String.valueOf(strings[i]));
69.                     // 在结果文件中添加空格分隔符（除最后一个数据外）
70.                     if (i < strings.length - 1) {
71.                         writer.write(" ");
72.                     }
73.                 }
74.                 // 在控制台和结果文件中换行
75.                 writer.write("\n");
76.                 writer.close();
77.             } catch (IOException e) {
78.                 e.printStackTrace();
79.             }
80.
81.         }
82.         reader.close();
83.     } catch (FileNotFoundException e) {
84.         e.printStackTrace();
85.     } catch (IOException e) {
86.         throw new RuntimeException(e);

```


4.运行结果展示:

为便于展示将排序后的数组存入两个新的 txt 文件中

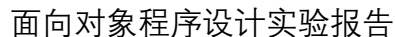
[illegible]

radiocontrol.txt - 文本

33209 42871 74841 95011 10090 116707 116069 140974 152995 175047 183885 220320 227820 249780 270338 282444 314751 379940 384703 414482 419210 425046 440760 452359 500928 522828 548134 556421 563203 568093 589636 667413 687294 719585 748033 753394
780243 831130 841534 886753 906300 979390 987071 1004168 1042056 1074011 1094687 1122375 1137502
21 48 124 324 418 492 677 1138 1296 1313 1710 2151 2243 2384 2530 2622 2664 3097 3113 3389 3419 3434 3528 3631 3777 3795 3852 3926 3934 4181 4374 4467 4558 4759 4938 5019 5024 5088 5294 5344 5542 5863 6065 6184 6239 6274 6399 6596 6983 7545 7763 17203
3391 33840 44930 59375 95333 18111 19476 161839 22281 231007 239820 25512 268308 324024 324407 330906 355032 359764 364422 382426 411570 454446 474102 507083 566144 570282 597162 618866 641190 668377 679557 695679 704304 709632 713089 786053
840908 876237 891849 911010 927315 943803 958208 1029940 1080774 1083889 1164499 1222416
146 436 439 724 843 909 937 1063 1213 1654 1767 1849 2036 2359 2742 2862 2876 3465 3732 3832 4181 4189 4197 4233 4282 4304 4403 4456 4715 4757 4796 5822 6019 6104 6110 6233 6641 6870 6880 6902 6916 7086 7110 7249 7256 7322 7490 7564 7869 7948 7955 36035
77176 103443 104704 117326 123284 138786 165515 22334 232084 246437 265525 290975 296803 305159 314577 323354 330312 350874 354953 362220 386135 408562 415497 470222 530773 533430 552397 562445 595356 600791 600936 66465 688958 690401 690848
756575 813963 822539 836954 872667 874072 903379 920788 97171 987601 1022150 1207500 1233807
128 528 813 936 1018 1159 1185 1189 1622 1912 1914 1944 2084 2427 2471 2500 2614 2846 2856 2901 2941 3006 3030 3237 3357 3546 3570 3662 4095 4190 4382 4489 4840 4857 4863 5791 5791 6012 6168 6412 6661 6778 6807 7035 7152 7199 7340 7439 7474 7476 7870
126487 158899 16555 17456 20180 204632 264105 288660 317090 326955 330088 344002 366614 378203 380554 401887 411740 421858 432970 443014 483513 487782 526499 526938 571874 637702 664951 697795 705591 739280 761461 789871 854537 855263 863837
86795 869423 904291 92598 92751 1003154 1033764 1048520 1130771 1163524 1195465 1214914 1222154 122800
140 147 552 617 695 720 838 1228 1239 1332 1523 1559 1665 1682 1697 1738 1771 1776 1860 2027 2099 2161 2233 2458 3003 3138 3267 3588 3747 3762 4165 4346 4810 5022 5062 5240 5259 5496 5547 5598 5744 5893 6182 6538 6869 7101 7162 7387 7668 7743 82416 98677
129508 137630 194614 199394 202216 225709 288432 307181 319758 323695 336235 351336 373028 380859 414294 416956 456511 472831 516642 521121 573760 585924 632771 652953 679689 701223 702234 747067 759709 765766 791517 795646 805638 808073
859148 889361 893777 939216 947301 974151 1046655 1071746 110058 1103131 1120876 1177414 1239176
265 457 466 875 1002 1533 1568 1981 2010 2070 2392 2465 2525 2569 2645 2707 2784 2848 3126 3143 3189 3278 3297 3300 3325 3329 3433 3504 3634 3681 3851 4382 4451 4593 4907 5006 5258 5870 5924 5932 6350 6406 6413 6578 6885 6924 7262 7553 7678 7994 53923
79514 101914 156133 168942 176777 224303 242526 260241 268057 270038 290006 290117 316514 383853 40009 445474 491340 516126 546623 550187 592996 631780 640672 644335 652766 656838 669474 687280 727064 753759 759648 772513 774174 784709 806098
831050 840195 872150 88446 875623 950502 961508 103203 1043876 1103650 1148689 1181843 1194653 122525
37 618 874 891 997 1155 1295 1441 1491 1516 1737 1743 1747 1765 1774 2078 2354 2373 2430 2480 2787 2806 3254 3449 3498 3601 3759 4005 4193 4350 4439 4558 4618 4943 4951 4956 5094 5312 5412 5500 5999 6257 6293 6411 6597 6631 6731 6842 6889 6904 7402 7488 7799 15269 19136
33658 118047 152734 166922 172605 282932 308927 310119 318018 319244 339204 366018 366102 398740 411102 428450 444666 457765 497148 506632 527289 544229 558933 616566 628893 659106 659946 711570 754530 777278 779800 790336 805162 825038 843182
895271 1035703 1046559 1052854 1080096 1092927 1093808 1099573 1124109 1161462 1203540 1204609 1225955
133 319 322 335 366 643 891 909 1127 1332 1793 1925 1996 2233 2430 2481 2787 2806 3254 3449 3498 3601 3759 4005 4193 4350 4439 4558 4618 4943 4951 4956 5094 5312 5412 5500 5999 6257 6293 6411 6597 6631 6731 6842 6889 6904 7402 7488 7799 15269 19136
21671 74917 88022 102422 121543 121847 141577 142157 144660 159487 172929 188624 205520 249296 252559 274774 278346 287658 290001 342267 431648 484579 490388 519761 557243 564164 580503 584607 586952 591950 652689 662689 706111 747625 768407 80564
917608 919840 924142 958053 973095 987071 990839 1006837 1019626 1031889 1087605 1219773
43 97 255 367 590 735 1224 1480 1539 1761 1925 2450 2882 3042 3045 3323 3427 3542 3936 3994 4014 4290 4412 4444 4466 4591 4823 4908 5117 5442 5493 5567 5574 5836 5874 5970 5975 6005 6075 6454 6589 6795 6856 7109 7345 7807 7811 7863 7944 7994 32781 36265
42599 45213 72754 92440 123273 144992 161915 187760 215182 22464 244167 277152 277784 278102 293645 296992 315570 374945 377022 396673 472585 490640 553919 593805 634228 637858 679556 679718 680477 683073 752529 761873 784037 867180 891016 950553
989195 1027688 1029705 1031562 1041353 1051762 110232 1103190 1116189 1147200 1164479 1199819
229 345 357 603 1054 1087 1622 1625 1786 2108 2146 2326 2391 2490 2614 2862 2936 3332 3391 3724 3769 3836 3974 4026 4359 4389 4461 4471 4852 4962 5051 5353 5372 5428 5458 5902 5964 6565 6870 6999 7036 7117 7233 7259 7267 7333 7620 7841 7873 7945 7995
18182 21342 38058 38512 46799 97951 109684 133520 180349 232259 253515 262503 342190 344974 360470 399585 429571 475720 543617 591235 607804 688715 717303 733955 758588 82698 856364 880872 883297 893649 895965 915233 934896 945564 96414 1045926
1048072 1053133 1063160 1080488 1089956 1104039 1134255 1144002 1129577 1139237 1170951 1195705 1221546
29380 37846 43295 66553 67026 117289 128677 143121 148175 188132 189378 193510 222111 330957 339443 375809 380557 398615 403828 421363 426930 43693 437124 527564 528292 534843 568431 583876 60033 631286 645624 748973 760970 766238 810649 840289
848554 850950 856877 880763 888043 943227 965153 977333 980540 997633 1016867 1183479
37 87 175 299 384 463 731 824 939 1162 1412 1735 1782 2054 2184 2245 2341 2418 2420 3146 3175 3451 3480 3672 3935 4000 4181 4261 4541 4550 4623 4631 5088 5114 5468 5544 5697 6030 6084 6534 6721 7016 7177 7244 7345 7417 7496 7368 7808 7821 16329 82762 90855
98131 101126 111201 121588 151194 202859 204814 209296 220218 231289 236708 303457 328296 331379 356688 361337 383226 388561 416427 436400 438222 467894 530104 542971 546314 561333 627743 653121 674038 685422 686164 727268 776489 874339 897223
929047 961300 995519 1002881 1008096 1032540 1044311 1051832 1076501 1080631 1153389 1203599
47 70 200 270 335 602 695 953 1363 1405 1619 1850 1870 1934 2201 2292 2952 3040 3041 3213 3477 3651 3882 4113 4174 4277 4344 4499 4671 4925 4953 5244 5504 5560 5706 5840 5941 5978 6034 6110 6852 6894 6943 7290 7322 7470 7504 7677 7680 7815 7968 16350 69722
98911 192701 225863 236912 262738 285437 327597 355745 371367 387453 444507 449023 477863 534517 557370 575360 597915 603461 605070 649376 678063 744998 749119 766606 772939 777631 780399 805966 872771 892161 918666 942360 958599 959382 961485
1002689 10054154 1007683 111105 1148364 1178092 1196882 1201257 1202775 1221077 122578 1228735 1228735
126 154 175 202 215 560 638 667 736 743 786 788 802 869 985 1114 1236 1439 1562 1981 2018 2367 2696 2883 2959 2997 3172 3192 3280 3413 3569 3615 3939 4001 4102 4208 4600 5200 5544 5559 6073 6406 6551 6639 6647 6680 6875 7285 7384 7511 7891 7995 68715 80261
108411 123355 147212 203642 233571 246067 250079 306935 309244 348963 391788 406930 459844 471968 507392 531104 554802 57796 580873 591462 598452 605641 619551 623880 630762 651793 661273 672150 685609 763585 763848 785023 816259 816963 818743
948885 963608 969484 982654 994091 999134 1045797 1047480 1072628 1086809 1150786
33 339 363 379 662 1077 1350 1682 1737 1776 2014 2070 2093 2215 2274 2311 2379 2444 2770 2918 3101 3177 3180 3229 3524 3915 4032 4507 4542 4587 4663 4740 4845 5239 5248 5358 5396 5628 5799 6035 6053 6190 6276 6277 6360 6575 6824 6927 7202 7257 7956 27719
84465 129712 170764 215549 25129 269001 287350 289515 295457 306718 309882 399712 423034 444570 476123 509940 558475 580563 609466 612573 616156 644541 695141 705352 718578 736314 746030 758445 759793 809546 871735 890247 928419 943083 949562
878571 882335 918095 939551 948123 980495 1021161 1024876 1051097 1068013 1203380 1222440
28 56 319 554 714 1000 1019 1132 1167 1262 1342 1463 1541 1986 2446 2642 3214 4168 4371 4482 4472 4507 4522 4569 4597 4844 4865 5083 5385 5655 5896 6055 6112 6173 6448 6542 6556 6663 6716 6765 6994 7056 7134 7378 7387 7650 7651 7913 15570 20388
52734 61852 83495 119603 137959 189467 192771 256264 260946 265138 287343 389129 311065 322968 365267 374877 376208 386416 444379 487348 518973 529406 533180 560587 583238 608138 652470 659976 662400 675710 699384 751128 779995 78083 794879 909439
95146 95452 95528 957669 966612 1021870 1038918 1080328 1083440 1083890 1162342 1214714
37 207 237 296 478 653 671 746 1033 1614 2164 2237 2325 2401 2540 2548 2766 2776 2855 2944 2972 3240 3275 3617 3773 4042 4145 4175 4188 4373 4506 4530 4863 4917 5145 5238 5225 5466 5578 5578 5755 6554 6554 6515 6871 6923 7369 7464 7474 7486 7803 26270 53436
69811 74895 116227 166705 167075 175516 256392 274463 390217 348954 355665 361536 397056 456302 457748 479786 480589 496237 567588 583347 595178 587011 638963 68043 728451 738089 763281 800864 827283 838674 840577 885330 918616 945581 948899
953134 954714 965743 1028034 1030482 1033317 1055184 1124240 1148910 1156861 1173378 1176164 1211032

显示行: 置 993 到 100% (Line Info) UTF-8

为了方便检验，编写了 isSorted 方法来判断所给数组是否有序，再次测试得到以下结果：



附录

任务一；

```

1. package task1;
2.
3. public class ListException extends Exception {
4.     public ListException(String message) {
5.         super(message);
6.     }
7.
8.     public ListException(String message, Throwable cause) {
9.         super(message, cause);
10.    }
11.
12.    public ListException(Throwable cause) {
13.        super(cause);
14.    }
15.}
16.

```

Ctrl+M

```

1.
2.     package task1;
3.
4.
5. import java.io.PrintWriter;
6.
7. public interface List<T> {
8.     void insert(T newElement) throws ListException;
9.     /*
10.      * Precondition: List is not full and newElement is not
11.      * null.
12.      * PostCondition:
13.      * Inserts newElement into a list after the cursor. If
14.      * the list is empty,newElement is inserted as the first(and o
15.      * nly)element in the list.
16.      * In either case(empty or not empty),moves the cursor
17.      * to newElement.
18.      * If there is not enough space in the list, throw a Li
19.      * stException exception.
20.      * ListException is a custom Exception class which you
21.      * should define it.
22.      */
23.     void remove() throws ListException;
24.     /*
25.      * Precondition:is

```



```

18.      * List is not empty.
19.      * PostCondition:
20.      * Removes the element marked by the cursor from a list
      .If the resulting list is not empty,
21.      * then moves the cursor to the element that followed t
      he deleted element.If the deleted element
22.      * was at the end of the list,then moves the cursor to
      the element at the beginning of the list.
23.      *
24.      */
25.      void replace(T newElement) throws ListException;
26.      /*
27.      * Precondition:
28.      * List is not empty and newElement is not null.
29.      * PostCondition:
30.      * Replaces the element marked by the cursor with newEl
      ement. The cursor remains at newElement.
31.      */
32.      void clear();
33.      /*
34.      * Precondition:
35.      * None
36.      * PostCondition:
37.      * Removes all the elements in a list.
38.      */
39.      boolean isEmpty();
40.      /*
41.      * Precondition:
42.      * None
43.      * PostCondition:
44.      * Returns true if a list is empty. Otherwise , returns
      false.
45.      */
46.      double getSpaceOccupancy();
47.      boolean isFull();
48.      /*
49.      * Precondition:
50.      * None
51.      * PostCondition:
52.      * Returns true if a list is full. Otherwise, returns f
      alse.
53.      */
54.      boolean gotoBeginning();
55.      /*
56.      * Precondition:

```

```

57.      * None
58.      * PostCondition:
59.      * If a list is not empty, then moves the cursor to the
      beginning of
60.      * the list and returns true.Otherwise,returns false.
61.      */
62.      boolean gotoEnd();
63.      /*
64.      * Precondition:
65.      * None
66.      * PostCondition:
67.      * If a list is not empty,then moves the cursor to the
      end of the list
68.      * and returns true.Otherwise, returns false.
69.      */
70.      boolean gotoNext();
71.      /*
72.      * Precondition:
73.      * List is not empty.
74.      * PostCondition:
75.      * If the cursor is not at the end of a list,then moves
      the cursor to the
76.      * next element in the list and return true.Otherwise,r
      eturns false.
77.      */
78.      boolean gotoPrev();
79.      /*
80.      * Precondition:
81.      * List is not empty.
82.      * PostCondition:
83.      * If the cursor is not at the beginning of a list,then
      moves the cursor to
84.      * the preceding element in the list and returns true.O
      therwise,returns false.
85.      */
86.      T getCursor();
87.      /*
88.      * Precondition:
89.      * List is not empty.
90.      * PostCondition:
91.      * Returns a copy of the element marked by the cursor.
92.      */
93.      void showStructure(PrintWriter pw);
94.      /*
95.      * Precondition:

```

```

96.      * None
97.      * PostCondition:
98.      * Outputs the elements in a list and the value of curs
      or. If the list is empty, outputs "Empty list".
99.      * Note that this operation is intended for testing/deb
      uggung purpose only.
100.     */
101.     void moveToNth(int n) throws ListException;
102.     /*
103.     * Precondition:
104.     * List contains at least n + 1 elements.
105.     * Postcondition:
106.     * Removes the element marked by the cursor from a l
      ist and reinserts it as the nth element in the list, where
      the elements are numbered from beginning to end, starting w
      ith zero. Moves the cursor to the moved element.
107.     */
108.     boolean find(T searchElement);
109.     /*
110.     * Precondition:
111.     * List is not empty.
112.     * Postcondition:
113.     * Searches a list for searchElement. Begins the sea
      rch with the element marked by the cursor. Moves the cursor
      through the list until either searchElement is found (retu
      rns true) or the end of the list is reached without finding
      searchElement (returns false). Leaves the cursor at the la
      st element visited during the search.
114.     */
115. }

```

Ctrl+M

```

1.
2. package task1;
3.
4. import java.io.PrintWriter;
5. import java.util.Arrays;
6.
7. public class ArrayListDemo implements List<Character>{
8.     private int capacity;
9.     private int length;
10.    private Character[] array;
11.    private int cursor;
12.    public ArrayListDemo(int capacity){
13.        this.capacity = capacity;

```

```

14.         cursor = -1;
15.         length=0;
16.         array = new Character[capacity];}
17.     public void insert(Character newElement) throws ListExc
        eption{
18.         if(isFull()){
19.             throw new ListException("List is full");
20.             // 数组满则抛出异常
21.         }
22.         else{
23.             for(int i=length;i>cursor+1;i--){
24.                 array[i] = array[i-1];
25.                 // 将后面的元素向后移动一位
26.             }
27.             array[++cursor] = newElement;
28.             // 将新元素插入到 cursor 的位置
29.             length++;
30.         }
31.     }
32.     public double getSpaceOccupancy() {
33.         return 1.0*length/capacity;
34.     }
35.     public void remove() {
36.         if(isEmpty()){
37.             System.out.println("List is empty");
38.             return;
39.         }
40.         for (int i = cursor; i < length - 1; i++) {
41.             array[i] = array[i + 1];
42.         }
43.         length--;
44.         if(length==0){
45.             cursor=-1;
46.         }else if(cursor==length){
47.             cursor=0;
48.         }
49.
50.     }
51.     public void replace(Character newElement) throws ListEx
        ception {
52.         if(isEmpty()){
53.             insert(newElement);
54.             return;
55.         }
56.         array[cursor] = newElement;

```



```

57.     }
58.     public void clear() {
59.         array=new Character[capacity];
60.         length = 0;
61.         cursor = -1;
62.     }
63.     public boolean isEmpty() {
64.         if(length==0){
65.             return true;
66.         }
67.         else{
68.             return false;
69.         }
70.     }
71.     public boolean isFull() {
72.         if(length==capacity){
73.             return true;
74.         }
75.         else{
76.             return false;
77.         }
78.     }
79.     public boolean gotoBeginning() {
80.
81.         if(length==0){
82.             return false;
83.         }
84.         else{
85.             cursor = 0;
86.             return true;
87.         }
88.     }
89.     public boolean gotoEnd() {
90.
91.         if(length==0){
92.             return false;
93.         }
94.         else{
95.             cursor = length-1;
96.             return true;
97.         }
98.     }
99.     public boolean gotoNext() {
100.         if(cursor==length-1){
101.             return false;

```

```

102.         }
103.         else{
104.             cursor++;
105.             return true;
106.         }
107.     }
108.     public boolean gotoPrev() {
109.         if(cursor==0|cursor==1){
110.             return false;
111.         }
112.         else{
113.             cursor--;
114.             return true;
115.         }
116.     }
117.     public Character getCursor() {
118.         return array[cursor];
119.     }
120.     public void showStructure(PrintWriter pw){
121.         if(length==0){
122.             pw.print("Empty List");
123.         }else {
124.             for (int i = 0; i < length; i++) {
125.                 pw.print(array[i]);
126.                 pw.print(" ");
127.             }
128.         }
129.         pw.println("{capacity="+this.capacity+",length="
+this.length+",cursor="+this.cursor+"}");
130.     }
131.     public void moveToNth(int n) {
132.         if(cursor+1<n){
133.             Character temp=array[cursor];
134.             for(int i=cursor;i<n-1;i++){
135.                 array[i] = array[i+1];
136.             }
137.             array[n-1] = temp;
138.         }
139.         else if(cursor+1>n){
140.             Character temp=array[cursor];
141.             for(int i=cursor;i>=n;i--){
142.                 array[i] = array[i-1];
143.             }
144.             array[n-1] = temp;
145.         }

```

```

146.     }
147.     public boolean find(Character searchElement) {
148.         for(int i=cursor;i<length;i++){
149.             if(array[i]==searchElement){
150.                 cursor = i;
151.                 return true;
152.             }
153.         }
154.         for(int i=0;i<cursor;i++){
155.             if(array[i]==searchElement){
156.                 cursor = i;
157.                 return true;
158.             }
159.         }
160.         return false;
161.     }
162.
163. }

```

Ctrl+M

```

1. package task1;
2. import java.io.PrintWriter;
3. public class LinkedList implements List<Character> {
4.     private class Node<T> {
5.         private T data;
6.         private Node<T> next;
7.
8.         public Node(T data) {
9.             this.data = data;
10.            next = null;
11.        }
12.        public T getData() {
13.            return data;
14.        }
15.        public void setData(T data) {
16.            this.data = data;
17.        }
18.        public Node<T> getNext() {
19.            return next;
20.        }
21.        public void setNext(Node<T> next) {
22.            this.next = next;
23.        }
24.    }
25.    private Node<Character> head;

```

```

26.     private Node<Character> cursor;
27.     private Node<Character> tail;
28.     private int capacity;
29.     private int length;
30.
31.     public LinkedList(int capacity) {
32.         this.capacity = capacity;
33.         head=new Node<Character>(' ');
34.         cursor=head;
35.         tail=null;
36.         length=0;
37.     }
38.     public double getSpaceOccupancy() {
39.         return 1.0*length/capacity;
40.     }
41.     public void insert(Character newElement) throws ListException{
42.         if(isFull()) {
43.             throw new ListException("List is full");
44.         } else {
45.             Node<Character> newNode = new Node<Character>(newElement);
46.             if(isEmpty()) {
47.                 head.setNext(newNode);
48.                 tail = newNode;
49.                 cursor = newNode;
50.             } else {
51.                 newNode.setNext(cursor.getNext());
52.                 cursor.setNext(newNode);
53.                 cursor=cursor.getNext();
54.                 if(cursor.getNext()==null){
55.                     tail=cursor;
56.                 }
57.             }
58.             length++;
59.         }
60.     }
61.     public void remove() throws ListException {
62.         if(isEmpty()) {
63.             throw new ListException("List is empty");
64.         }
65.         Node<Character> temp=head;
66.         while(true){
67.             if(temp.getNext()==cursor){
68.                 break;

```

```

69.         }else{
70.             temp=temp.getNext();
71.         }
72.     }
73.     temp.setNext(cursor.getNext());
74.     if(isEmpty()){
75.         clear();
76.         return;
77.     }else if(cursor.getNext()==null){
78.         tail=temp;
79.         cursor=head.getNext();
80.     }else cursor=temp.next;
81.     length--;
82. }
83. public void replace(Character newElement) throws ListEx
    ception {
84.     Node<Character> newElementNode=new Node<Character>(
        newElement);
85.     Node<Character> temp=head;
86.     if(cursor==head){
87.         insert(newElement);
88.         return;
89.     }
90.     while(true){
91.         if(temp.getNext()==cursor){
92.             break;
93.         }else{
94.             temp=temp.getNext();
95.         }
96.     }
97.     newElementNode.next=cursor.getNext();
98.     if(cursor.getNext()==null){
99.         tail=newElementNode;
100.    };
101.    cursor=newElementNode;
102.    temp.setNext(newElementNode);
103. }
104. public void clear() {
105.     head.setNext(null);
106.     tail=null;
107.     length=0;
108.     cursor=head;
109. }
110. public boolean isEmpty() {
111.     if(head.getNext()==null) {

```

```

112.         return true;
113.     } else {
114.         return false;
115.     }
116. }
117. public boolean isFull() {
118.     if(length==capacity) {
119.         return true;
120.     } else {
121.         return false;
122.     }
123. }
124. public boolean gotoBeginning() {
125.     if(head.getNext()==null){
126.         return false;
127.     }else{
128.         cursor=head.getNext();
129.         return true;
130.     }
131. }
132. public boolean gotoEnd() {
133.     if(head.getNext()==null){
134.         return false;
135.     }else{
136.         cursor=tail;
137.         return true;
138.     }
139. }
140. public boolean gotoNext() {
141.     if(cursor.getNext()==null){
142.         return false;
143.     }else{
144.         cursor=cursor.getNext();
145.         return true;
146.     }
147. }
148. public boolean gotoPrev() {
149.     if(cursor==head.getNext()||cursor==head){
150.         return false;
151.     }else{
152.         Node<Character> temp=head;
153.         while(true){
154.             if(temp.getNext()==cursor){
155.                 break;
156.             }else{

```



```

157.             temp=temp.getNext();
158.         }
159.     }
160.     cursor=temp;
161.     return true;
162. }
163. }
164. public Character getCursor(){
165.     return cursor.getData();
166. }
167. public int getcursorIndex(){
168.     Node<Character> temp=head;
169.     int index=-1;
170.     while(true){
171.         if(temp==cursor){
172.             break;
173.         }else{
174.             temp=temp.getNext();
175.             index++;
176.         }
177.     }
178.     return index;
179. }
180. public void showStructure(PrintWriter pw) {
181.     Node<Character> temp=head.getNext();
182.     while(temp!=null){
183.         pw.print(temp.getData());
184.         pw.print(" ");
185.         temp=temp.getNext();
186.     }
187.     pw.println("{"+"capacity="+capacity+", length="+
        length+", cursor="+getcursorIndex()+"}");
188. }
189. public void moveToNth(int n) {
190.     Node<Character> temp=head;
191.     while(true){
192.         if(temp.getNext()==cursor){
193.             break;
194.         }else{
195.             temp=temp.getNext();
196.         }
197.     }
198.     temp.setNext(cursor.getNext());
199.     temp=head;
200.     for(int i=0;i<n-1;i++){

```

```

201.         temp=temp.getNext();
202.     }
203.     cursor.setNext(temp.getNext());
204.     temp.setNext(cursor);
205. }
206. public boolean find(Character searchElement) {
207.     if(isEmpty()){
208.         return false;
209.     }else{
210.         Node<Character> temp=cursor;
211.         while(temp!=tail){
212.             if(temp.getNext().getData()==searchElement){
213.                 cursor=temp.getNext();
214.                 return true;
215.             }else{
216.                 temp=temp.getNext();
217.             }
218.         }
219.         temp=head;
220.         while(temp!=cursor){
221.             if(temp.getNext().getData()==searchElement){
222.                 cursor=temp.getNext();
223.                 return true;
224.             }else{
225.                 temp=temp.getNext();
226.             }
227.         }
228.         return false;
229.     }
230. }
231.
232. }
233.

```

Ctrl+M

```

1. package task1;
2. import java.io.PrintWriter;
3. public class TwowayLinkedList implements List<Character> {
4.     private class Node<T> {
5.         private T data;
6.         private Node<T> next;
7.         private Node<T> prev;
8.         public Node(T data) {

```

```

9.         this.data = data;
10.        next = null;
11.        prev = null;
12.    }
13.    public T getData() {
14.        return data;
15.    }
16.    public void setData(T data) {
17.        this.data = data;
18.    }
19.    public Node<T> getNext() {
20.        return next;
21.    }
22.    public void setNext(Node<T> next) {
23.        this.next = next;
24.    }
25.    public Node<T> getPrev() {
26.        return prev;
27.    }
28.    public void setPrev(Node<T> prev) {
29.        this.prev = prev;
30.    }
31. }
32. private Node<Character> head;
33. private Node<Character> cursor;
34. private Node<Character> tail;
35. private int capacity;
36. private int length;
37.
38. public TwowayLinkedList(int capacity) {
39.     this.capacity = capacity;
40.     head=new Node<Character>(' ');
41.     cursor=head;
42.     tail=null;
43.     length=0;
44. }
45. public void insert(Character newElement) throws ListExc
    eption{
46.     if(isFull()) {
47.         throw new ListException("List is full");
48.     } else {
49.         Node<Character> newNode = new Node<Character>(n
            ewElement);
50.         if(isEmpty()) {
51.             head.setNext(newNode);

```

```

52.         tail = newNode;
53.         cursor = newNode;
54.     }
55.     else if(cursor==head){
56.         newNode.setNext(head.getNext());
57.         head.getNext().setPrev(newNode);
58.         head.setNext(newNode);
59.         cursor=newNode;
60.     }else if(cursor==tail){
61.         cursor.setNext(newNode);
62.         newNode.setPrev(cursor);
63.         cursor=newNode;
64.         tail=newNode;
65.     }
66.     else {
67.         newNode.setNext(cursor.getNext());
68.         cursor.getNext().setPrev(newNode);
69.         newNode.setPrev(cursor);
70.         cursor.setNext(newNode);
71.         cursor=cursor.getNext();
72.     }
73.     length++;
74. }
75. }
76. public void remove() throws ListException {
77.     if(isEmpty()) {
78.         throw new ListException("List is empty");
79.     }
80.     if(cursor==tail&&cursor.getPrev()==head){
81.         clear();
82.         return;
83.     }
84.     if(cursor==tail) {
85.         cursor.getPrev().setNext(null);
86.         tail=cursor.getPrev();
87.         cursor=head.getNext();
88.     }else if(cursor==head.getNext()) {
89.         head.setNext(cursor.getNext());
90.         cursor=head.getNext();
91.     }else{
92.         cursor.getPrev().setNext(cursor.getNext());
93.         cursor.getNext().setPrev(cursor.getPrev());
94.         cursor=cursor.getNext();
95.     }
96.     length--;

```

```

97.     }
98.     public void replace(Character newElement) throws ListEx
        ception {
99.         Node<Character> newElementNode=new Node<Character>(
            newElement);;
100.         if(cursor==head||isEmpty()){
101.             insert(newElement);
102.             return;
103.         }
104.         if(cursor==tail&&cursor.getPrev()==head){
105.             newElementNode.setNext(cursor.getNext());
106.             cursor.getNext().setPrev(newElementNode);
107.             head.setNext(newElementNode);
108.         }else if(cursor==tail){
109.             cursor.getPrev().setNext(newElementNode);
110.             newElementNode.setPrev(cursor.getPrev());
111.             tail=newElementNode;
112.         }else if(cursor==head.getNext()){
113.             newElementNode.setNext(cursor.getNext());
114.             cursor.getNext().setPrev(newElementNode);
115.             head.setNext(newElementNode);
116.         }else{
117.             newElementNode.setNext(cursor.getNext());
118.             cursor.getNext().setPrev(newElementNode);
119.             newElementNode.setPrev(cursor.getPrev());
120.             cursor.getPrev().setNext(newElementNode);
121.         }
122.         cursor=newElementNode;
123.     }
124.     public void clear() {
125.         head.setNext(null);
126.         tail=null;
127.         length=0;
128.         cursor=head;
129.     }
130.     public boolean isEmpty() {
131.         if(head.getNext()==null) {
132.             return true;
133.         } else {
134.             return false;
135.         }
136.     }
137.     public boolean isFull() {
138.         if(length==capacity) {
139.             return true;

```

```

140.         } else {
141.             return false;
142.         }
143.     }
144.     public boolean gotoBeginning() {
145.         if(head.getNext()==null){
146.             return false;
147.         }else{
148.             cursor=head.getNext();
149.             return true;
150.         }
151.     }
152.     public boolean gotoEnd() {
153.         if(head.getNext()==null){
154.             return false;
155.         }else{
156.             cursor=tail;
157.             return true;
158.         }
159.     }
160.     public boolean gotoNext() {
161.         if(cursor.getNext()==null){
162.             return false;
163.         }else{
164.             cursor=cursor.getNext();
165.             return true;
166.         }
167.     }
168.     public boolean gotoPrev() {
169.         if(cursor==head.getNext()||cursor==head){
170.             return false;
171.         }else{
172.             cursor=cursor.getPrev();
173.             return true;
174.         }
175.     }
176.     public Character getCursor(){
177.         return cursor.getData();
178.     }
179.     public int getcursorIndex(){
180.         Node<Character> temp=head;
181.         int index=-1;
182.         while(true){
183.             if(temp==cursor){
184.                 break;

```



```

185.         }else{
186.             temp=temp.getNext();
187.             index++;
188.         }
189.     }
190.     return index;
191. }
192. public void showStructure(PrintWriter pw) {
193.     Node<Character> temp=head.getNext();
194.     while(temp!=null){
195.         pw.print(temp.getData());
196.         pw.print(" ");
197.         temp=temp.getNext();
198.     }
199.     pw.println("{"+"capacity="+capacity+", length="+
        length+", cursor="+getcursorIndex()+"}");
200. }
201. public void moveToNth(int n) {
202.     Node<Character> temp=head;
203.     while(true){
204.         if(temp.getNext()==cursor){
205.             break;
206.         }else{
207.             temp=temp.getNext();
208.         }
209.     }
210.     temp.setNext(cursor.getNext());
211.     temp=head;
212.     for(int i=0;i<n-1;i++){
213.         temp=temp.getNext();
214.     }
215.     cursor.setNext(temp.getNext());
216.     temp.setNext(cursor);
217. }
218. public double getSpaceOccupancy() {
219.     return 1.0*length/capacity;
220. }
221. public boolean find(Character searchElement) {
222.     if(isEmpty()){
223.         return false;
224.     }else{
225.         Node<Character> temp=cursor;
226.         while(temp!=tail){
227.             if(temp.getNext().getData()==searchElement){

```

```

228.             cursor=temp.getNext();
229.             return true;
230.         }else{
231.             temp=temp.getNext();
232.         }
233.     }
234.     temp=head;
235.     while(temp!=cursor){
236.         if(temp.getNext().getData()==searchElement){
237.             cursor=temp.getNext();
238.             return true;
239.         }else{
240.             temp=temp.getNext();
241.         }
242.     }
243.     return false;
244. }
245. }
246.
247. }
248.
249.

```

Ctrl+M

```

1. package task1;
2. import java.io.*;
3.
4. public class ArrayTestDemo {
5.     public static void main(String[] args) throws ListException{
6.         List<Character> list = new ArrayListDemo(512);
7.         //定义List 对象指向不同的类
8.         PrintWriter pw = new PrintWriter(System.out);
9.         int N=0;
10.        try (BufferedReader br = new BufferedReader(new FileReader("D:reportsource/list_testcase.txt"))) {
11.            String line;
12.            while ((line = br.readLine()) != null) {
13.                // 在这里处理每一行的文本
14.                execute(line, list);
15.                list.showStructure(pw);
16.            }
17.        } catch (IOException e) {
18.            e.printStackTrace();

```

```

19.     }
20.     pw.close();
21. }
22. public static void execute(String example, List<Character> list) throws ListException {
23.
24.     int i=0;
25.     while(i<example.length()){
26.         char a = example.charAt(i);
27.         switch (a) {
28.             case '+':
29.                 char insertchar = example.charAt(++i);
30.                 //插入元素，需要多读入一位字符，下面
                 replace 同理
31.                 try{list.insert(insertchar);
32.                 }catch (ListException e){
33.                     System.out.println("List is full");
34.                 }
35.                 break;
36.             case '-':
37.                 list.remove();
38.                 break;
39.             case '=':
40.                 char replacetchar = example.charAt(++i);
41.                 list.replace(replacetchar);
42.                 break;
43.             case '#':
44.                 list.gotoBeginning();
45.                 break;
46.             case '*':
47.                 list.gotoEnd();
48.                 break;
49.             case '>':
50.                 list.gotoNext();
51.                 break;
52.             case '<':
53.                 list.gotoPrev();
54.                 break;
55.             case '~':
56.                 list.clear();
57.                 break;
58.         }
59.
60.         i+=2;
61.     }

```

```
62.    }
63.}
64.
```

Ctrl+M

任务二：

```
1. package task2;
2. import task1.List;
3. import task1.ListException;
4.
5. import java.io.PrintWriter;
6. import java.util.Arrays;
7.
8. public class ResizingAlist implements List<Character> {
9.     private int capacity;
10.    private int length;
11.    private Character[] array;
12.    private int cursor;
13.    public ResizingAlist() {
14.        capacity = 1;
15.        cursor = -1;
16.        length = 0;
17.        array = new Character[capacity];
18.        Arrays.fill(array, null);
19.    }
20.
21.    public void insert(Character newElement) throws ListException {
22.        if(isFull()){
23.            // 数组满则扩容
24.            resize(capacity*2);
25.        }
26.        for(int i=length;i>cursor+1;i--){
27.            array[i] = array[i-1];
28.        }
29.        array[++cursor] = newElement;
30.        length++;
31.    }
32.
33.
34.    public void remove() {
35.        if(isEmpty()||cursor==-1){
36.            System.out.println("List is empty");
37.            return;
38.        }
39.    }
40.}
```

```

39.         for (int i = cursor; i < length - 1; i++) {
40.             array[i] = array[i + 1];
41.         }
42.         array[length-1] = null;
43.         length--;
44.         if(length==0){
45.             cursor=-1;
46.         }else if(cursor==length){
47.             cursor=0;
48.         }
49.         //扩容
50.         if(length>0&&length==capacity/4){
51.             resize(capacity/2);
52.         }
53.     }
54.     public double getSpaceOccupancy() {
55.         return 1.0*length/capacity;
56.     }
57.     public void replace(Character newElement) throws ListEx
        ception {
58.         if(cursor== -1){
59.             /*cursor++;
60.             array[cursor] = newElement;
61.             length++;*/
62.             insert(newElement);
63.             return;
64.         }
65.         array[cursor] = newElement;
66.     }
67.     public void clear() {
68.         Arrays.fill(array, null);
69.         length = 0;
70.         cursor = -1;
71.         capacity = 1;
72.     }
73.     public boolean isEmpty() {
74.         if(length==0){
75.             return true;
76.         }
77.         else{
78.             return false;
79.         }
80.     }
81.     public boolean isFull() {
82.         if(length==capacity){

```

```

83.
84.         return true;
85.     }
86.     else{
87.
88.         return false;
89.     }
90. }
91. public boolean gotoBeginning() {
92.
93.     if(length==0){
94.         return false;
95.     }
96.     else{
97.         cursor = 0;
98.         return true;
99.     }
100. }
101. public boolean gotoEnd() {
102.
103.     if(length==0){
104.         return false;
105.     }
106.     else{
107.         cursor = length-1;
108.         return true;
109.     }
110. }
111. public boolean gotoNext() {
112.     if(cursor==length-1||cursor==
113.         1||cursor==length){
114.         return false;
115.     }
116.     else{
117.         cursor++;
118.         return true;
119.     }
120. }
121. public boolean gotoPrev() {
122.     if(cursor==0||cursor==-1){
123.         return false;
124.     }
125.     else{
126.         cursor--;
127.         return true;

```



```

127.         }
128.     }
129.     public Character getCursor() {
130.         return array[cursor];
131.     }
132.     public void showStructure(PrintWriter pw){
133.         if(length==0){
134.             pw.print("Empty List");
135.         }else {
136.             for (int i = 0; i < length; i++) {
137.                 pw.print(array[i]);
138.                 pw.print(" ");
139.             }
140.         }
141.         pw.println("{capacity="+this.capacity+",length="+
            +this.length+",cursor="+this.cursor+"}");
142.     }
143.     public void moveToNth(int n) {
144.         if(cursor+1<n){
145.             Character temp=array[cursor];
146.             for(int i=cursor;i<n-1;i++){
147.                 array[i] = array[i+1];
148.             }
149.             array[n-1] = temp;
150.         }
151.         else if(cursor+1>n){
152.             Character temp=array[cursor];
153.             for(int i=cursor;i>=n;i--){
154.                 array[i] = array[i-1];
155.             }
156.             array[n-1] = temp;
157.         }
158.     }
159.     public boolean find(Character searchElement) {
160.         for(int i=cursor;i<length;i++){
161.             if(array[i]==searchElement){
162.                 cursor = i;
163.                 return true;
164.             }
165.         }
166.         for(int i=0;i<cursor;i++){
167.             if(array[i]==searchElement){
168.                 cursor = i;
169.                 return true;
170.             }

```

```

171.         }
172.         return false;
173.     }
174.     private void resize(int newlength){
175.         //新建长度数组
176.         Character[] newarray = new Character[newlength];
177.         //将原数组元素拷贝进新数组
178.         for(int i=0;i<length;i++){
179.             newarray[i] = array[i];
180.         }
181.         //将新数组赋给原数组
182.         array = newarray;
183.         //更新容量
184.         capacity = newlength;
185.     }
186.
187. }

```

Ctrl+M

```

1. package task1;
2.
3. import org.jfree.chart.ChartFactory;
4. import org.jfree.chart.ChartPanel;
5. import org.jfree.chart.JFreeChart;
6. import org.jfree.chart.plot.PlotOrientation;
7. import org.jfree.chart.plot.XYPlot;
8. import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
9. import org.jfree.chart.ui.ApplicationFrame;
10. import org.jfree.chart.ui.RectangleInsets;
11. import org.jfree.data.xy.XYDataset;
12. import org.jfree.data.xy.XYSeries;
13. import org.jfree.data.xy.XYSeriesCollection;
14. import task2.ResizingAlist;
15.
16. import java.awt.*;
17. import java.io.BufferedReader;
18. import java.io.FileReader;
19. import java.io.IOException;
20. import java.io.PrintWriter;
21.
22. public class LineXYDemo extends ApplicationFrame {
23.     // 该构造方法中完成了数据集、图表对象和显示图表面板的创建工作
24.     public LineXYDemo(String title){
25.         super(title);

```

```

26.         XYDataset dataset = createDataset();           //
           创建记录图中坐标点的数据集
27.         JFreeChart chart = createChart(dataset);       //
           使用上一步已经创建好的数据集生成一个图表对象
28.         ChartPanel chartPanel = new ChartPanel(chart);  //
           将上一步已经创建好的图表对象放置到一个可以显示的Panel 上
29.         // 设置GUI 面板Panel 的显示大小
30.         chartPanel.setPreferredSize(new Dimension(500, 270)
           );
31.         setContentPane(chartPanel);                     //
           这是JavaGUI 的步骤之一，不用过于关心，面向对象课程综合训练的视
           频中进行了讲解。
32.     }
33.
34.     private JFreeChart createChart(XYDataset dataset) {
35.         // 使用已经创建好的dataset 生成图表对象
36.         // JFreechart 提供了多种类型的图表对象，本次实验是需要使
           用XYLine 型的图表对象
37.         JFreeChart chart = ChartFactory.createXYLineChart(
38.             "Comparison of two ADT Implement ways",
           // 图表的标题
39.             "Operated lines",
           // 横轴的标题名
40.             "Space Occupancy",
           // 纵轴的标题名
41.             dataset,                                     // 图表对象中
           使用的数据集对象
42.             PlotOrientation.VERTICAL,                  // 图表显示的
           方向
43.             true,                                       // 是否显示图
           例
44.             false,                                     // 是否需要生
           成 tooltips
45.             false                                       // 是否需要生
           成 urls
46.         );
47.         // 下面所做的工作都是可选操作，主要是为了调整图表显示的
           风格
48.         // 同学们不必在意下面的代码
49.         // 可以将下面的代码去掉对比一下显示的不同效果
50.         chart.setBackgroundPaint(Color.WHITE);
51.         XYPlot plot = (XYPlot)chart.getPlot();
52.         plot.setBackgroundPaint(Color.lightGray);
53.         plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.
           0, 6.0));

```

```

54.         plot.setDomainGridlinePaint(Color.WHITE);
55.         plot.setRangeGridlinePaint(Color.WHITE);
56.         XYLineAndShapeRenderer renderer = (XYLineAndShapeRe
           nderer) plot.getRenderer();
57.         renderer.setDefaultShapesVisible(true);
58.         renderer.setDefaultShapesFilled(true);
59.         return chart;
60.     }
61.
62.     private XYDataset createDataset() {
63.         double[] X = new double[125];
64.         for(int i = 0; i < 125; i++) {
65.             X[i] = i;
66.         }
67.         double[] Y1 =new double[125];
68.         double[] Y2 =new double[125];
69.         int m=0,n=0;
70.         List<Character> list1 = new ArrayListDemo(512);
71.         List<Character> list2 = new ResizingAlist();
72.         PrintWriter pw = new PrintWriter(System.out);
73.         try (BufferedReader br = new BufferedReader(new Fil
           eReader("D:reportsource/list_testcase.txt"))) {
74.             String line;
75.             while ((line = br.readLine()) != null) {
76.                 // 在这里处理每一行的文本
77.                 execute(line, list1);
78.                 Y1[m++] = list1.getSpaceOccupancy();
79.             }
80.         } catch (ListException | IOException e) {
81.             e.printStackTrace();
82.         }
83.         try (BufferedReader br = new BufferedReader(new Fil
           eReader("D:reportsource/list_testcase.txt"))) {
84.             String line;
85.             while ((line = br.readLine()) != null) {
86.                 // 在这里处理每一行的文本
87.                 execute(line, list2);
88.                 Y2[n++] = list2.getSpaceOccupancy();
89.             }
90.         } catch (IOException | ListException e) {
91.             e.printStackTrace();
92.         }
93.         pw.close();
94.         double[][] Y = {Y1, Y2};

```

```

95.      // jfreechart 中使用XYSeries 对象存储一组数据的(x,y)的
        序列, 因为有三组数据所以创建三个XYSeries 对象
96.      XYSeries[] series = {new XYSeries("Fixed ArrayImpl"
        ), new XYSeries("Resizing ArrayImpl")};
97.      int N = X.length;
98.      int M = series.length;
99.      for(int i = 0; i < M; i++)
100.         for(int j = 0; j < N; j++)
101.            series[i].add(X[j], Y[i][j]);
102.      // 因为在该图表中显示的数据序列不止一组, 所以在
        jfreechart 中需要将多组数据序列存放到一个XYSeriesCollection 对
        象中
103.      XYSeriesCollection dataset = new XYSeriesCollect
        ion();
104.      for(int i = 0; i < M; i++)
105.         dataset.addSeries(series[i]);
106.      return dataset;
107.  }
108.  public static void execute(String example, List<Char
        acter> list) throws ListException {
109.
110.      int i=0;
111.      while(i<example.length()){
112.          char a = example.charAt(i);
113.          switch (a) {
114.              case '+':
115.                  char insertchar = example.charAt(++i
                    );
116.                  try{list.insert(insertchar);
117.                  }catch (ListException e){
118.                      System.out.println("List is full
                        ");
119.                  }
120.                  break;
121.              case '-':
122.                  list.remove();
123.                  break;
124.              case '=':
125.                  char replacetchar = example.charAt(+
                    +i);
126.                  list.replace(replacetchar);
127.                  break;
128.              case '#':
129.                  list.gotoBeginning();
130.                  break;

```

```

131.         case '*':
132.             list.gotoEnd();
133.             break;
134.         case '>':
135.             list.gotoNext();
136.             break;
137.         case '<':
138.             list.gotoPrev();
139.             break;
140.         case '~':
141.             list.clear();
142.             break;
143.     }
144.
145.         i+=2;
146.     }
147. }
148. public static void main(String[] args) {
149.     LineXYDemo demo = new LineXYDemo("Comparison of
    two ADT Implement ways");
150.     demo.pack();
151.     demo.setVisible(true);
152. }
153. }
154.

```

Ctrl+M

任务三：

```

1. package task3;
2.
3. import java.util.Stack;
4.
5. public class Calculator {
6.     public static double calculate(String expression) throw
    s UnequalException {
7.         if (isMatch(expression)) {
8.             Stack<Double> numStack = new Stack<>();
9.             Stack<Character> opStack = new Stack<>();
10.
11.             for (int i = 0; i < expression.length(); i++) {
12.                 char c = expression.charAt(i);
13.                 if (Character.isDigit(c) || c == '.') {
14.                     StringBuilder sb = new StringBuilder();

```



```

15.         while (i < expression.length() && (Char
            acter.isDigit(expression.charAt(i)) || expression.charAt(i)
                == '.')) {
16.             sb.append(expression.charAt(i++));
17.         }
18.         i--;
19.         numStack.push(Double.parseDouble(sb.toS
            tring()));
20.     } else if (c == '(') {
21.         opStack.push(c);
22.     } else if (c == ')') {
23.         while (opStack.peek() != '(') {
24.             performOperation(numStack, opStack);
25.         }
26.         opStack.pop();
27.     } else if (c == '+' || c == '-'
        ' || c == '*' || c == '/' || c == '^') {
28.         while (!opStack.isEmpty() && precedence
            (opStack.peek()) >= precedence(c)) {
29.             performOperation(numStack, opStack);
30.         }
31.         opStack.push(c);
32.     }
33. }
34.
35. while (!opStack.isEmpty()) {
36.     performOperation(numStack, opStack);
37. }
38.
39. return numStack.pop();
40. }else{
41.     throw new UnequalException("The number of ( and
        ) in this expression are not equal!");
42. }
43. }
44. public static boolean isMatch(String str) {
45.     Stack<Character> stack = new Stack<>();
46.
47.     for (char c : str.toCharArray()) {
48.         if (c == '(') {
49.             // 左括号 入栈
50.             stack.push(c);
51.         } else if (c == ')') {
52.             // 右括号需要与栈顶左括号配对
53.             if (stack.empty() || stack.pop() != '(') {

```

```

54.             return false;
55.         }
56.     }
57. }
58.
59.     // 最后检查栈是否为空
60.     return stack.empty();
61. }
62. private static void performOperation(Stack<Double> numS
    tack, Stack<Character> opStack) {
63.     char op = opStack.pop();
64.     double num2 = numStack.pop();
65.     double num1 = numStack.pop();
66.     double result = 0;
67.     switch(op) {
68.         case '+':
69.             result = num1 + num2;
70.             break;
71.         case '-':
72.             result = num1 - num2;
73.             break;
74.         case '*':
75.             result = num1 * num2;
76.             break;
77.         case '/':
78.             result = num1 / num2;
79.             break;
80.         case '^':
81.             result = Math.pow(num1, num2);
82.             break;
83.     }
84.     numStack.push(result);
85. }
86.
87. private static int precedence(char op) {
88.     if (op == '+' || op == '-') {
89.         return 1;
90.     } else if (op == '*' || op == '/') {
91.         return 2;
92.     } else if (op == '^') {
93.         return 3;
94.     } else {
95.         return 0;
96.     }
97. }

```

```
98.}
99.
```

Ctrl+M

```
1. package task3;
2.
3. import java.util.Scanner;
4.
5. public class CalculatorTest {
6.     public static void main(String[] args){
7.         Scanner scanner = new Scanner(System.in);
8.         System.out.printf("请输入表达式:");
9.         String expression = scanner.nextLine();
10.        try{
11.            double result = Calculator.calculate(expression
12.        );
13.            System.out.println("计算结果为: " + result);
14.        }catch (UnequalException e){
15.            System.out.println(e.getMessage());
16.        }
17.    }
18.}
```

Ctrl+M

```
1. package task3;
2. public class LeakyStack<T> {
3.     private T[] stack;
4.     private int top;
5.     private int bottom;
6.     private int capacity;
7.     private int size;
8.
9.     public LeakyStack(int capacity) {
10.        this.capacity = capacity;
11.        stack = (T[]) new Object[capacity];
12.        top = 0;
13.        bottom = 0;
14.        size = 0;
15.    }
16.
17.    public void push(T item) {
18.        if (size == capacity) {
19.            // 栈已满, 需要"泄漏"最旧的元素
20.            bottom = (bottom + 1) % capacity;
```

```

21.         } else {
22.             size++;
23.         }
24.         //将item入栈
25.         stack[top] = item;
26.         //注意循环数组中bottom和top的更新方式
27.         top = (top + 1) % capacity;
28.     }
29.
30.     public T pop() {
31.         if (isEmpty()) {
32.             return null;
33.         }
34.         //注意循环数组中bottom和top的更新方式
35.         top = (top - 1 + capacity) % capacity;
36.         size--;
37.         //将栈顶元素出栈
38.         return stack[top];
39.     }
40.
41.     public boolean isEmpty() {
42.         return size == 0;
43.     }
44.     public boolean isFull() {
45.         return size == capacity;
46.     }
47.     public int getsize() {
48.         return size;
49.     }
50.     public void show() {
51.         System.out.print("Stack: ");
52.         if (isEmpty()) {
53.             System.out.println("Empty");
54.         } else {
55.             for (int i = 0; i < size; i++) {
56.                 System.out.print(stack[(bottom+i)%capacity]
57.                     + " ");
58.             }
59.             System.out.println();
60.         }
61.
62.     }

```

```

1. package task3;
2.
3. import java.util.Scanner;
4.
5. public class LeakyStackTest {
6.     public static void main(String[] args) {
7.         Scanner scanner = new Scanner(System.in);
8.         System.out.printf("请输入最大容量:");
9.         int capacity = scanner.nextInt();
10.        LeakyStack<Integer> leakystack = new LeakyStack<>(c
            apacity);
11.        while (true) {
12.            System.out.printf("输入需要操作:");
13.            String input = scanner.next();
14.            if (input.equals("show")) {
15.                leakystack.show();
16.            } else if (input.equals("pop")) {
17.                System.out.println(leakystack.pop());
18.            } else if (input.equals("push")) {
19.                System.out.printf("请输入元素:");
20.                int element = scanner.nextInt();
21.                leakystack.push(element);
22.            } else if (input.equals("exit")) {
23.                break;
24.            }
25.        }
26.        scanner.close();
27.    }
28. }
29.

```

Ctrl+M

```

1. package task3;
2.
3. import java.util.Stack;
4.
5. public class QuickSort2 {
6.
7.     static void quickSort(int[] arr) {
8.         Stack<Integer> stack = new Stack<>();
9.         stack.push(0);
10.        stack.push(arr.length);
11.
12.        while (!stack.isEmpty()) {
13.            int end = stack.pop();

```

```

14.         int start = stack.pop();
15.         if (end - start < 2) {
16.             continue;
17.         }
18.
19.         int p = start + ((end - start) / 2);
20.         p = partition(arr, p, start, end);
21.
22.         stack.push(p + 1);
23.         stack.push(end);
24.
25.         stack.push(start);
26.         stack.push(p);
27.     }
28. }
29.
30. private static int partition(int[] arr, int position, i
    nt start, int end) {
31.     int l = start;
32.     int h = end - 2;
33.     int piv = arr[position];
34.     swap(arr, position, end - 1);
35.
36.     while (l < h) {
37.         if (arr[l] < piv) {
38.             l++;
39.         } else if (arr[h] >= piv) {
40.             h--;
41.         } else {
42.             swap(arr, l, h);
43.         }
44.     }
45.     int idx = h;
46.     if (arr[h] < piv) {
47.         idx++;
48.     }
49.     swap(arr, end - 1, idx);
50.     return idx;
51. }
52.
53. private static void swap(int[] arr, int i, int j) {
54.     int temp = arr[i];
55.     arr[i] = arr[j];
56.     arr[j] = temp;
57. }

```

58. }

Ctrl+M

```
1. package task3;
2.
3. import java.util.Scanner;
4.
5. public class QuickSortTest {
6.     public static void main(String[] args) {
7.         Scanner scanner = new Scanner(System.in);
8.         System.out.println("请输入排序数组大小:");
9.         int size = scanner.nextInt();
10.        int[] arr = new int[size];
11.        System.out.println("请输入排序数组:");
12.        for (int i = 0; i < size; i++) {
13.            arr[i] = scanner.nextInt();
14.        }
15.        QuickSort2.quickSort(arr);
16.        System.out.println("排序结果为:");
17.        for (int i = 0; i < size; i++) {
18.            System.out.print(arr[i] + " ");
19.        }
20.    }
21. }
22. }
```

Ctrl+M

```
1. package task3;
2.
3. public class UnequalException extends Exception{
4.     public UnequalException(String message) {
5.         super(message);
6.     }
7. }
8. }
```

Ctrl+M

任务四:

```
1. package task4;
2. import java.util.LinkedList;
3.
4. public class Queue<T> {
5.     //private LinkedList<T> list = new LinkedList<>();
6.     private T[] list;
7.     private int size;
```



```

8.     private int head;
9.     private int tail;
10.    public Queue() {
11.        this(100);
12.    }
13.    public Queue(int size) {
14.        list = (T[]) new Object[size];
15.        this.size = size;
16.        head = -1;
17.        tail = 0;
18.    }
19.
20.    public void enqueue(T item) {
21.
22.        if (isFull()) {
23.            throw new RuntimeException("Queue is full");
24.        }
25.        if (this.head == -1) {
26.            this.head = 0;
27.        }
28.        list[tail] = item;
29.        tail++;
30.
31.    }
32.
33.    public T dequeue() {
34.
35.        if (isEmpty()) {
36.            throw new RuntimeException("Queue is empty");
37.        } else {
38.            T item = list[head];
39.            list[head] = null;
40.            head++;
41.            return item;
42.        }
43.    }
44.
45.    public boolean isEmpty() {
46.
47.        return this.head == -1 || head == tail;
48.    }
49.    public boolean isFull() {
50.
51.        return this.tail == this.size;
52.    }

```

```

53.
54.     public int getSize() {
55.         return size;
56.     }
57.     public void show(){
58.         for (int i = head; i < tail; i++) {
59.             System.out.print(list[i] + " ");
60.         }
61.     }
62. }
63.

```

Ctrl+M

```

1. package task4;
2.
3. import java.io.*;
4. import java.util.ArrayList;
5. import java.io.BufferedReader;
6. import java.io.FileReader;
7. import java.io.IOException;
8. import java.util.ArrayList;
9.
10. import java.util.ArrayList;
11. import java.util.Arrays;
12.
13. public class RadixSort {
14.     public static void radixSort(int[] arr) {
15.         // 找出最大数的位数
16.         int maxNumber = findMaxNumber(arr);
17.         int maxLength = Integer.toString(maxNumber).length(
18.         );
19.         // 对每一位进行排序
20.         for (int digit = 0; digit < maxLength; digit++) {
21.             // 使用队列数组作为桶
22.             Queue<Integer>[] buckets = new Queue[10];
23.             for (int i = 0; i < buckets.length; i++) {
24.                 buckets[i] = new Queue<>();
25.             }
26.
27.             // 按当前位数分配到桶中
28.             for (int number : arr) {
29.                 int bucketIndex = (number / (int) Math.pow(
30.                 10, digit)) % 10;
31.                 buckets[bucketIndex].enqueue(number);
32.             }
33.             // 按桶的顺序将数字放回数组
34.             int index = 0;
35.             for (Queue<Integer> bucket : buckets) {
36.                 while (!bucket.isEmpty()) {
37.                     arr[index++] = bucket.poll();
38.                 }
39.             }
40.         }
41.     }
42.
43.     private static int findMaxNumber(int[] arr) {
44.         int max = 0;
45.         for (int num : arr) {
46.             if (num > max) {
47.                 max = num;
48.             }
49.         }
50.         return max;
51.     }
52. }

```

```

31.         }
32.
33.         // 从桶中收集数字
34.         int arrayIndex = 0;
35.         for (Queue<Integer> bucket : buckets) {
36.             while (!bucket.isEmpty()) {
37.                 arr[arrayIndex++] = bucket.dequeue();
38.             }
39.         }
40.     }
41. }
42. public static void radixSort(String[] arr) {
43.     int maxLength = arr[0].length();
44.
45.     for (int digit = maxLength - 1; digit >= 0; digit--) {
46.         Queue<String>[] buckets = new Queue[52]; // 26
           letters in the alphabet
47.         for (int i = 0; i < buckets.length; i++) {
48.             buckets[i] = new Queue<>();
49.         }
50.
51.         for (String s : arr) {
52.             int bucketIndex;
53.             if(s.charAt(digit) >= 'A' && s.charAt(digit)
           ) <= 'Z'){
54.                 bucketIndex = s.charAt(digit) - 'A';
55.             }else{
56.                 bucketIndex = s.charAt(digit) -
           'a' + 26;
57.             }
58.             buckets[bucketIndex].enqueue(s);
59.         }
60.
61.         int arrayIndex = 0;
62.         for (Queue<String> bucket : buckets) {
63.             while (!bucket.isEmpty()) {
64.                 arr[arrayIndex++] = bucket.dequeue();
65.             }
66.         }
67.     }
68. }
69.
70. private static int findMaxNumber(int[] arr) {
71.     int max = arr[0];

```

```

72.         for (int i = 1; i < arr.length; i++) {
73.             if (arr[i] > max) {
74.                 max = arr[i];
75.             }
76.         }
77.         return max;
78.     }
79. }
80.

```

Ctrl+M

```

1. package task4;
2. import java.io.*;
3. import java.util.Scanner;
4.
5. public class SortTestdemo {
6.     public static boolean isSorted(int[] arr) {
7.         for (int i = 0; i < arr.length - 1; i++) {
8.             if (arr[i] > arr[i+1]) {
9.                 return false;
10.            }
11.        }
12.        return true;
13.    }
14.    public static boolean isSorted(String[] arr) {
15.        for (int i = 0; i < arr.length - 1; i++) {
16.
17.            if (arr[i].compareTo(arr[i+1]) > 0) {
18.                return false;
19.            }
20.        }
21.        return true;
22.    }
23.    public static void main(String[] args) {
24.        /*try {
25.
26.            File file = new File("D:/reportsource/radixSort
27.                1.txt");
28.            Scanner scanner = new Scanner(file);
29.            File outputFile = new File("D:/reportsource/rad
30.                ixSortresult1.txt");
31.            while (scanner.hasNextLine()) {
32.
33.                String line = scanner.nextLine();
34.                String[] dataArray = line.split(" ");
35.
36.                int[] arr = new int[dataArray.length];
37.                for (int i = 0; i < dataArray.length; i++) {
38.                    arr[i] = Integer.parseInt(dataArray[i]);
39.                }
40.                isSorted(arr);
41.            }
42.        } catch (IOException e) {
43.            e.printStackTrace();
44.        }
45.    }
46. }

```

```

33.         int[] numbers = new int[dataArray.length];
34.         for (int i = 0; i < dataArray.length; i++)
35.         {
36.             numbers[i] = Integer.parseInt(dataArray
37. [i]);
38.         }
39.         RadixSort.radixSort(numbers);
40.         try {
41.             FileWriter writer = new FileWriter(outp
42. utFile, true);
43.             for (int i = 0; i < numbers.length; i++
44. ) {
45.                 writer.write(String.valueOf(num
46. bers[i]));
47.                 if (i < numbers.length - 1) {
48.                     writer.write(" ");
49.                 }
50.             }
51.             System.out.println(isSorted(numbers));
52.             writer.write("\n");
53.             writer.close();
54.         } catch (IOException e) {
55.             e.printStackTrace();
56.         }
57.     }
58.     scanner.close();
59. } catch (FileNotFoundException e) {
60.     e.printStackTrace();
61. }*/
62. try {
63.     BufferedReader reader = new BufferedReader(new
64. FileReader("D:/reportsource/radixSort2.txt"));
65.     File outputFile2 = new File("D:/reportsource/ra
66. dixSortresult2.txt");
67.     String line;
68.     while ((line=reader.readLine()) != null) {
69.         String[] dataArray2 = line.split(" ");
70.         String[] strings = dataArray2;
71.         RadixSort.radixSort(strings);
72.         try {
73.             FileWriter writer = new FileWriter(outp
74. utFile2, true);

```

```

70.         for (int i = 0; i < strings.length; i++
       ) {
71.             writer.write(String.valueOf(strings
               [i]));
72.             if (i < strings.length - 1) {
73.                 writer.write(" ");
74.             }
75.         }
76.         System.out.println(isSorted(strings));
77.         writer.write("\n");
78.         writer.close();
79.     } catch (IOException e) {
80.         e.printStackTrace();
81.     }
82.
83.     }
84.     reader.close();
85. } catch (FileNotFoundException e) {
86.     e.printStackTrace();
87. } catch (IOException e) {
88.     throw new RuntimeException(e);
89. }
90. }
91. }
92.

```

Ctrl+M