时长：25h

## 欢迎学习《玩转数据结构》

### 欢迎学习《玩转数据结构》

### 学习数据结构(和算法)到底有没有用？

### 关于课程学习的更多注意事项

### 课程编程环境搭建

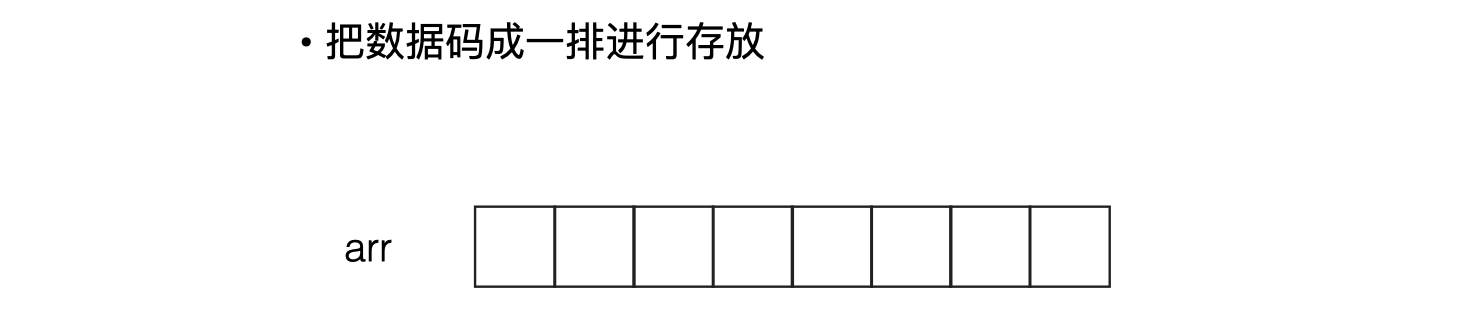
### 关于课程的其他语言支持(Python,C++,JS,Go)

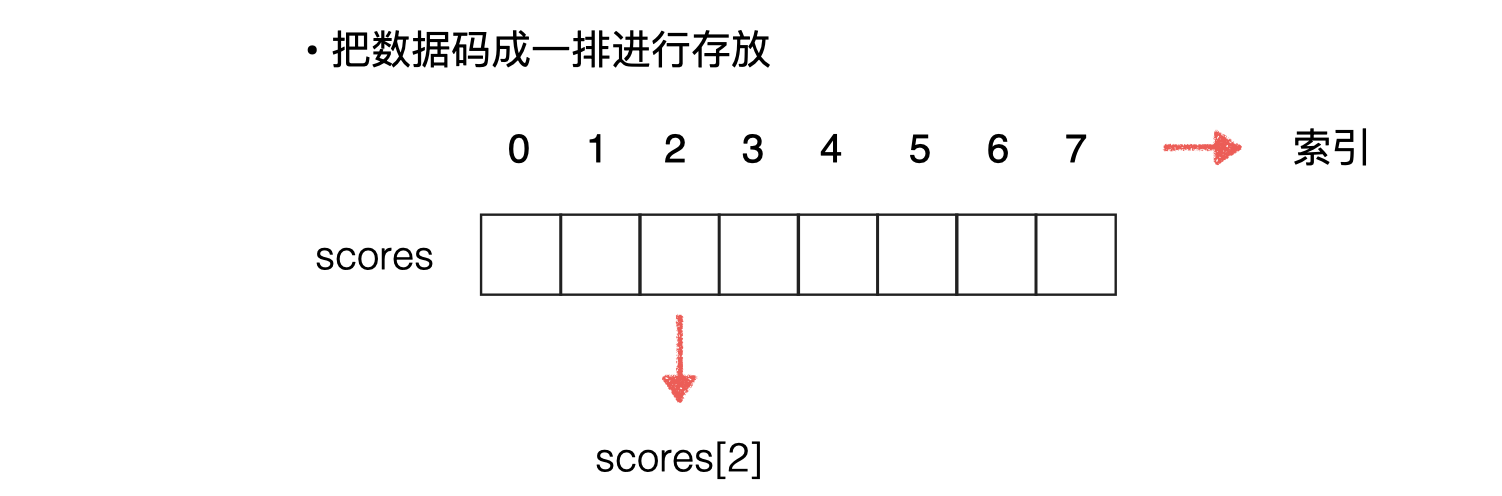
### 在学习数据结构的具体知识前，你可能想读一读这两篇文章

## 不要小瞧数组

### 使用 Java 中的数组

》》先复习下数组的基础知识





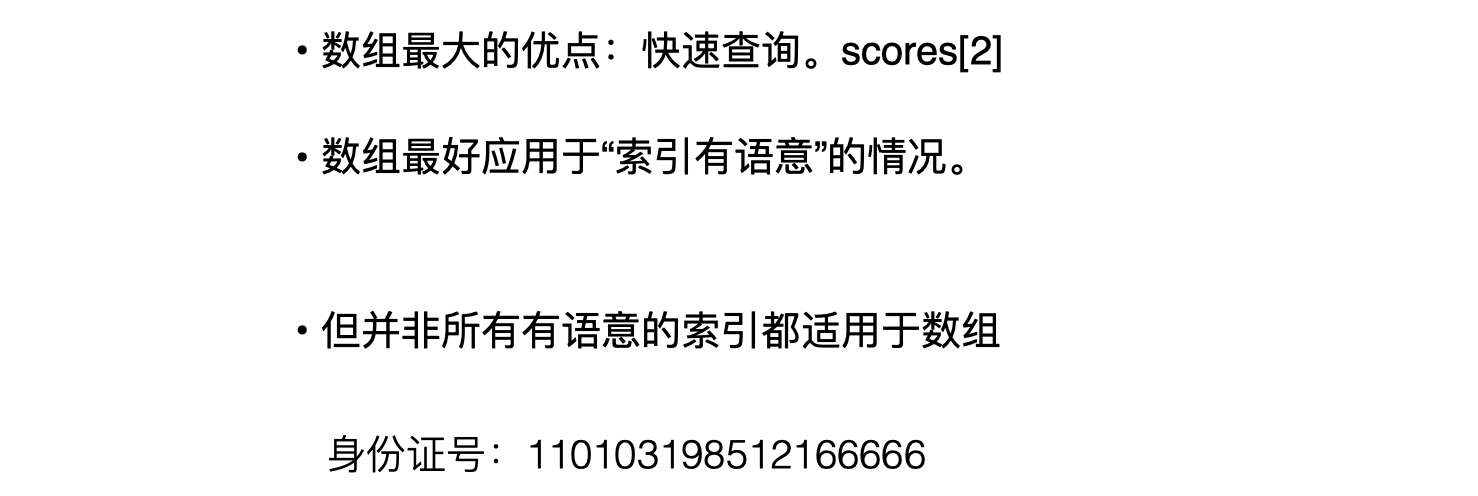
### 二次封装属于我们自己的数组

》》这里对数组进行封装，用我们的意义来进行

A screenshot of a cell phone

Description automatically generated

这里对数据的索引进行一下解释，数组既可以有语义也可以不用有语义

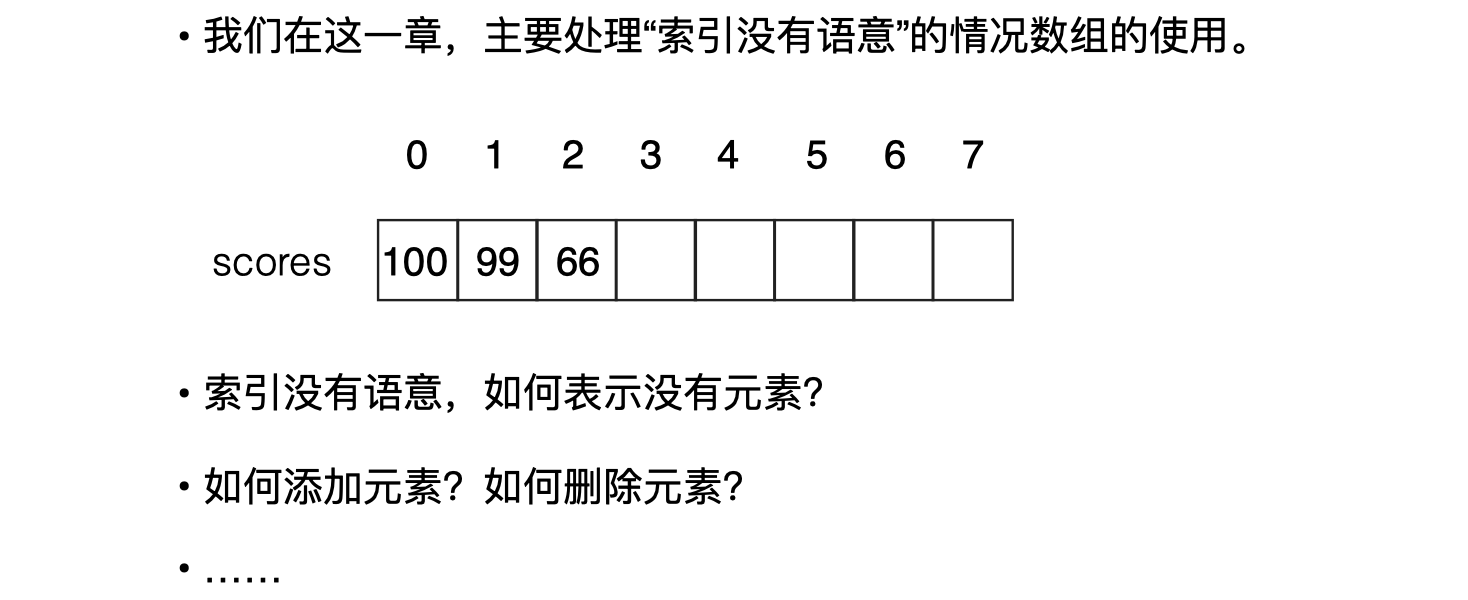


在本章中，我们使用的数组都是没有语意的

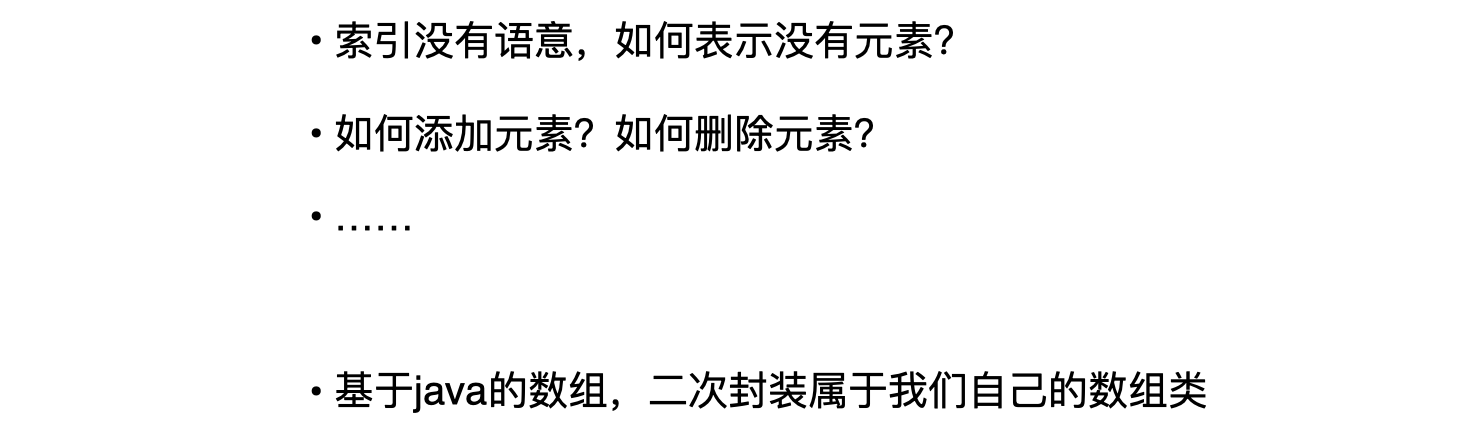
A screenshot of a cell phone

Description automatically generated

那既然没有意义，下面的问题如何处理了?



答案就是我们**封装自己的数组**



具体的我们的数组会有如下的方法

A screenshot of a cell phone

Description automatically generated

》》 下面具体编写代码

|  |
| --- |
| public class Array {  private int[] data; // 数据  private int size; // 多少个元素    private static final int DEFAULT\_CAPACITY = 10;    public Array(int capacity) {  this.data = new int[capacity];  this.size = 0;  }  public Array() {  this(DEFAULT\_CAPACITY);  }  // 获取数组元素个数  public int size(){  return this.size;  }  // 获取数组容量  public int getCapacity() {  return this.data.length;  }  // 数组是否为空  public boolean isEmpty() {  return this.size == 0;  }  } |

### 向数组中添加元素

这节课像我们的数组中添加元素，分为**向末尾添加元素**、向**任意位置添加元素**

**void addFirst(int item); // 数组首部添加元素**

**void adddLast(int item); // 数组尾部添加元素**

**void add(int index, int item); // 任意位置添加元素**

》》首先是在**数组末尾添加元素**

A screenshot of a cell phone

Description automatically generated

\*\* 代码的编写

|  |
| --- |
| void addLast(int item) {  data[size] = item;  size++;  } |

》》然后是在**任意位置添加元素**

中间的截图如下

A screenshot of a cell phone

Description automatically generated

\*\* 代码的编写

// 是否需要先检查容量

|  |
| --- |
| void add(int index, int item){  for(int i = size; i >= index ; i --) {  data[i] = data[i-1];  }  data[index] = item;  } |

### 数组中查询元素和修改元素

编写代码

在数组中查询元素和修改元素，这个通过序号就可以解决了

**int get(int index); // 获取元素**

**voie set(int index, int item); // 更新元素**

代码如下：

|  |
| --- |
| int get(int index){  return data[index];  }  int set(int index, int item) {  data[index] = item;  } |

### 包含、搜索和删除

具体要实现的代码是：

**boolean contains(int item); // 是否包含元素**

**int find(int item;); // 查找元素所在的索引位置**

**void remove(int index); // 移除 index 位置的元素**

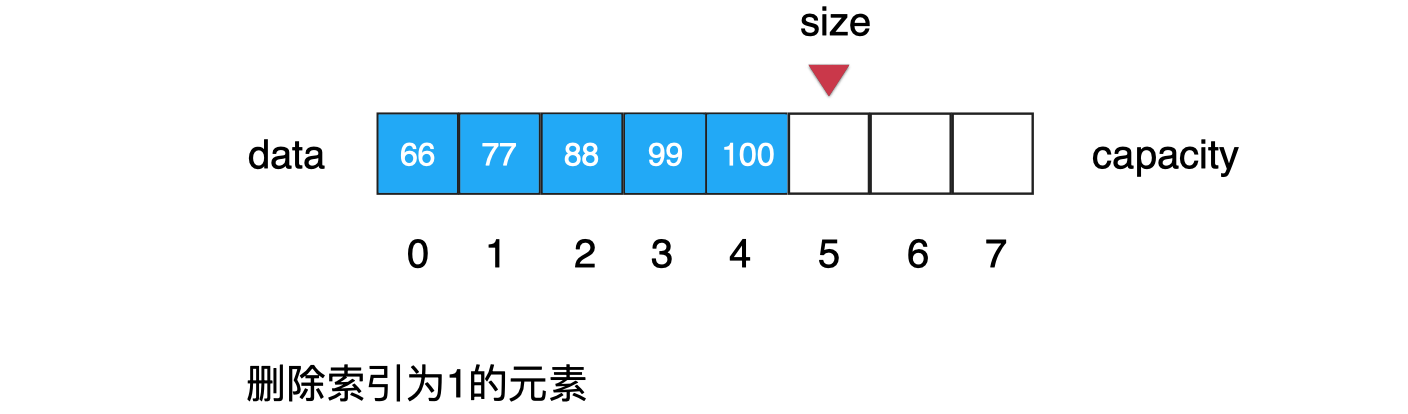
**void removeFirst();**

**void removeLast();**

》》从数组中删除元素

删除指定位置元素

如下图就是，删除 77 元素，删除完以后，需要将后面的元素一个一个的往前移，最后别忘了 size--。



具体代码如下

|  |
| --- |
| public boolean contais(int item){  for(int i = 0;i < size;i++) {  if(data[i] == item) {  return true;  }  }  return false;  }  public int find (int item){  for(int i = 0;i < size;i++) {  if(data[i] == item) {  return i;  }  }  return 0;  }  public void remove(int index) {  for(int i = index; i < size; i++) {  data[i] = data[i + 1];  }  data[size] = 0;  } |

### 使用泛型

上面的数组支持 int 类型，这里我们进行扩展，让数组可以支持的类型更多

》》如下图所示

A screenshot of a cell phone

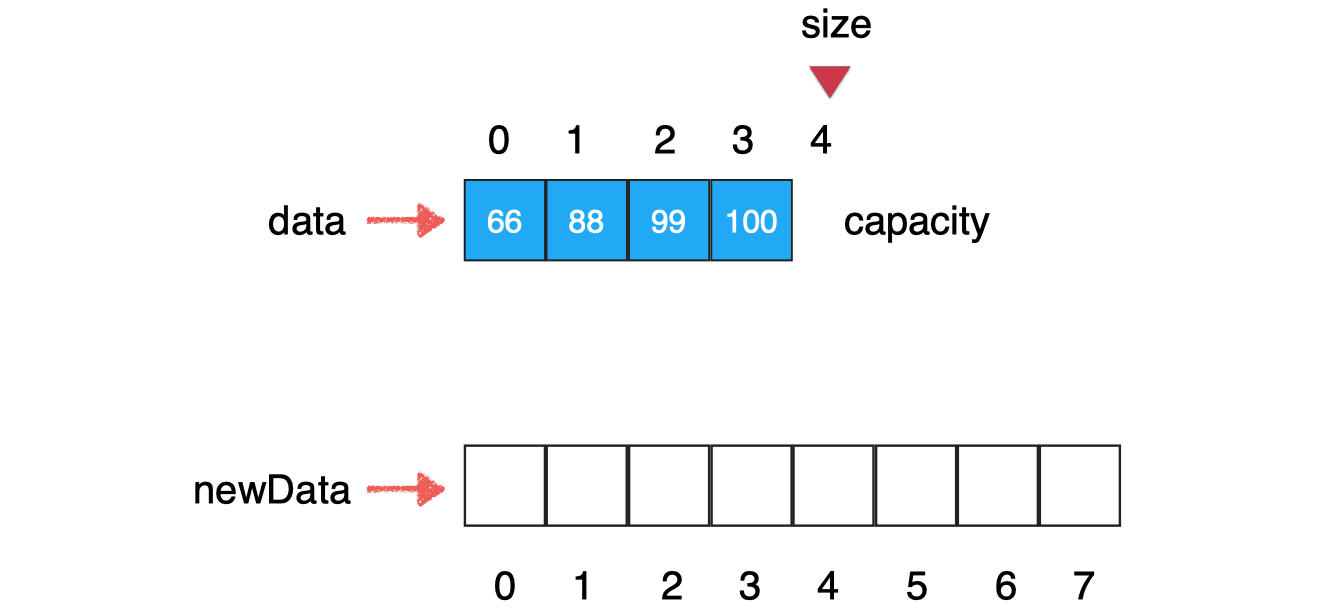
Description automatically generated

》》修改上面的代码，让我们的数组支持泛型

|  |
| --- |
|  |

### 动态数组

》》看下图

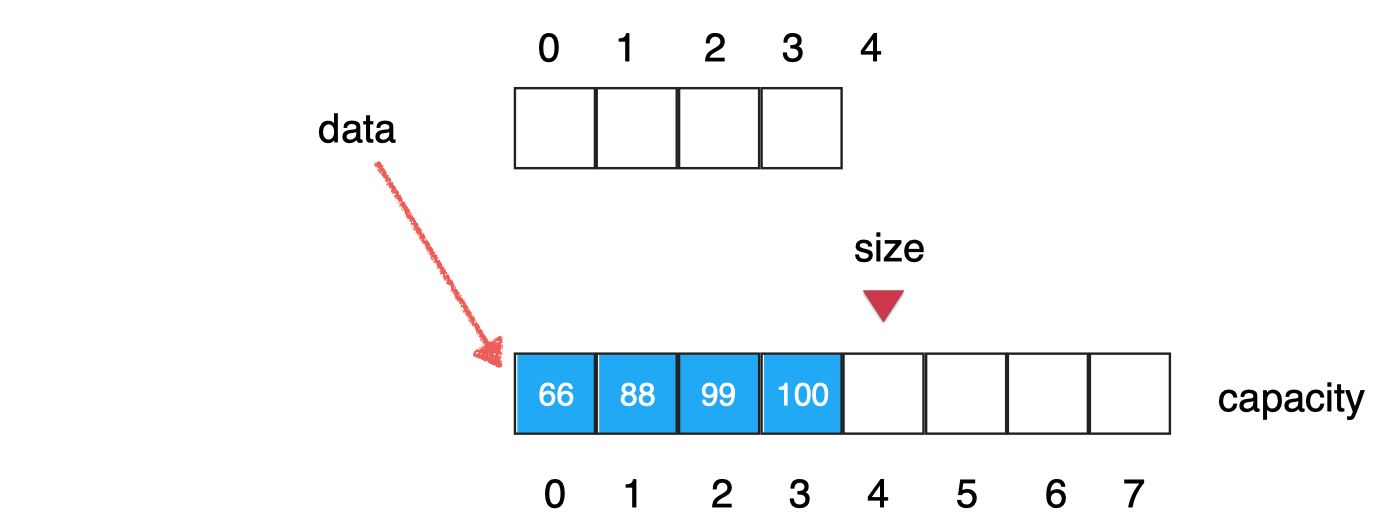


data 到 newdata 的就相当于数组扩容操作

A screenshot of a cell phone

Description automatically generated

最后将 data指向新的数组，就扩容了数组，就相当于动态数组



》》下面进行动态数组的代码的编写

|  |
| --- |
|  |

### 简单的复杂度分析

》》先介绍下时间复杂度的概念

A screenshot of a cell phone

Description automatically generated

》》为什么实用大 O 的方法来了？

A screenshot of a cell phone

Description automatically generated

》》下面看几个具体的例子

A screenshot of a cell phone

Description automatically generated

》》下面分析下动态数组的时间复杂度

A screenshot of a cell phone

Description automatically generated

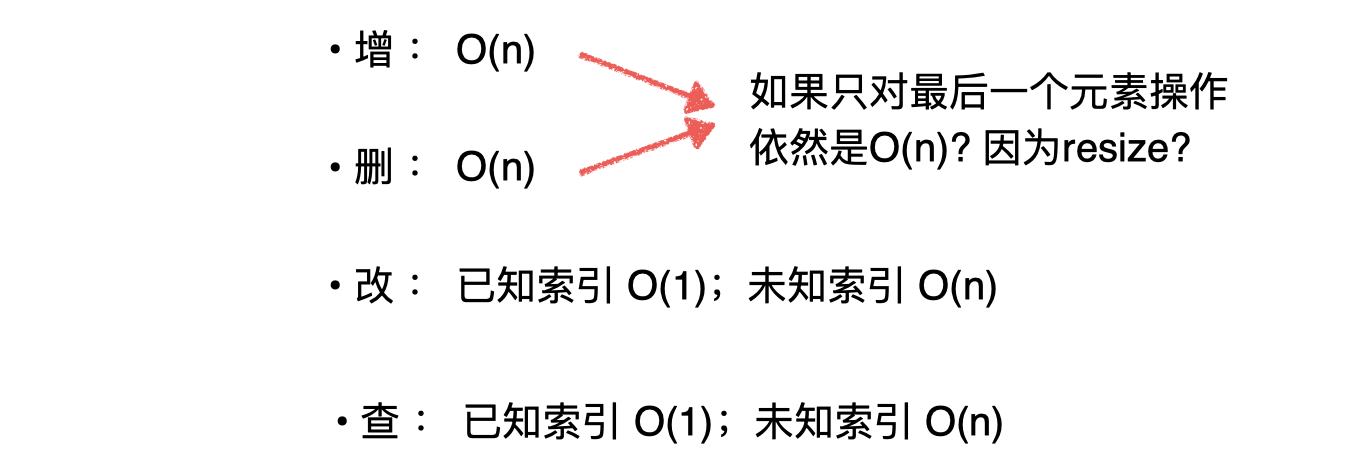
A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated





### 均摊复杂度和防止复杂度的震荡

》》这里我们进行resize 的复杂度分析

A picture containing object

Description automatically generated

A screenshot of a cell phone

Description automatically generated

\*\* 举例说明：

A screenshot of a cell phone

Description automatically generated

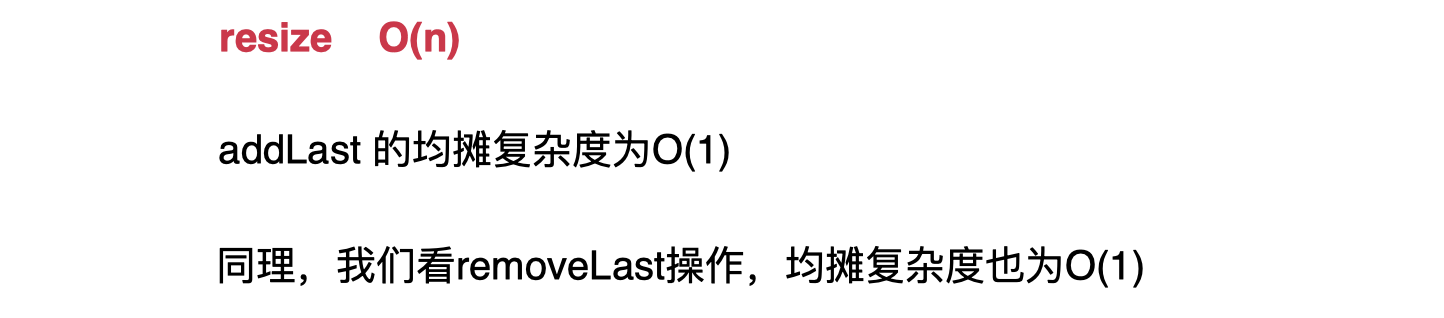
A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

》》均摊复杂度(amortized time complexity)



》》复杂度震荡

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

》》这里改造代码防止复杂度震荡

## 栈和队列

### 栈和栈的应用：撤销操作和系统栈

### 栈的基本实现

### 栈的另一个应用：括号匹配

### 关于 LeetCode 的更多说明

### 数组队列

### 循环队列

### 循环队列的实现

### 数组队列和循环队列的比较

## 最基础的动态数据结构:链表

### 什么是链表

### 在链表中添加元素

### 使用链表的虚拟头结点

### 链表的遍历，查询和修改

### 从链表中删除元素

### 使用链表实现栈

### 带有尾指针的链表：使用链表实现队列

## 链表和递归

### Leetcode 中和链表相关的问题

### 测试自己的 Leetcode 链表

### 递归基础与递归的宏观语意

### 链表的天然递归结构性质

### 递归运行的机制：递归的微观解读

### 递归算法的调试

### 更多和链表相关的问题

## 二分搜索树

## 集合和映射

## 优先队列和堆

## 线段树

## Trie

## 并查集

## AVL

## 红黑树

## 哈希表

## 结尾语