数据结构与算法

面试篇

1. 基础知识

2. 数组

3. 字符串

4. 链表

5. 队列

6. 堆栈

基础知识

- 程序 =算法 + 数据结构
- 算法：就是解决问题的方法或者过程
- 数据结构：是数据的计算机表示和相应的一组操作

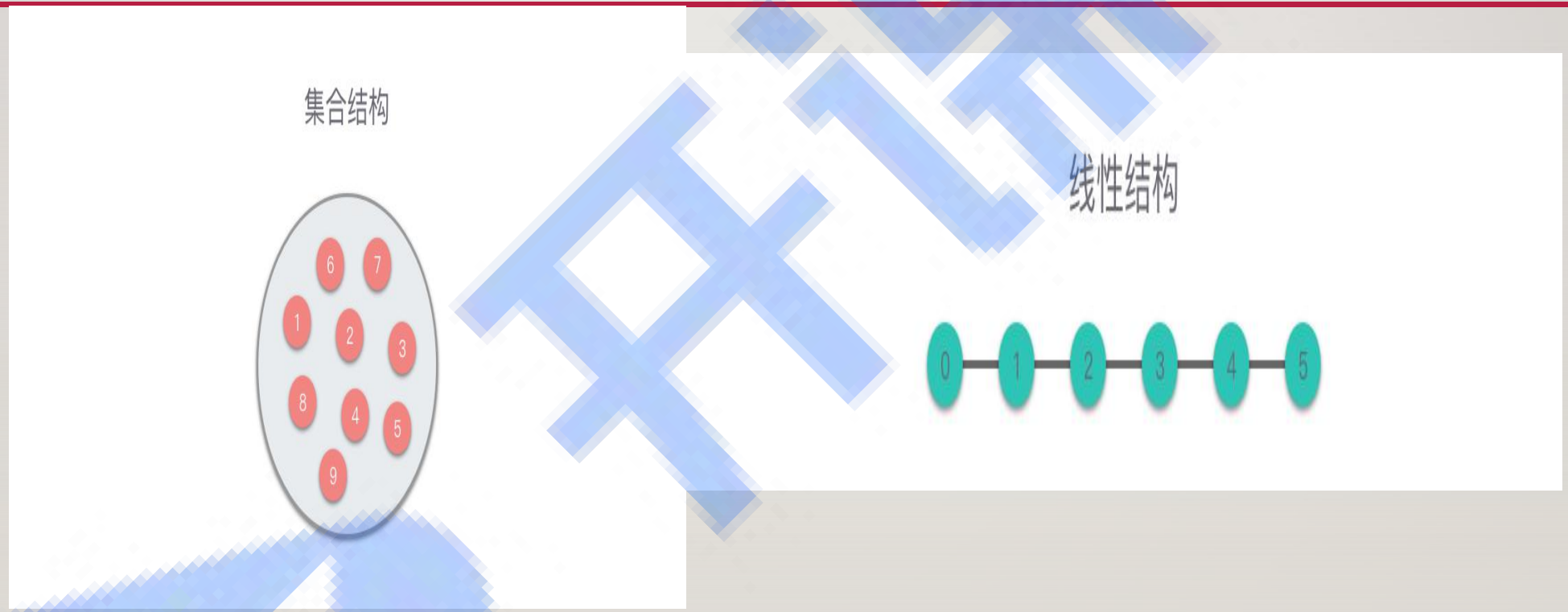
数据结构

- **逻辑结构**
  - 数据元素之间的相互关系。
- **物理结构**
  - 数据的逻辑结构在计算机中的存储方式。

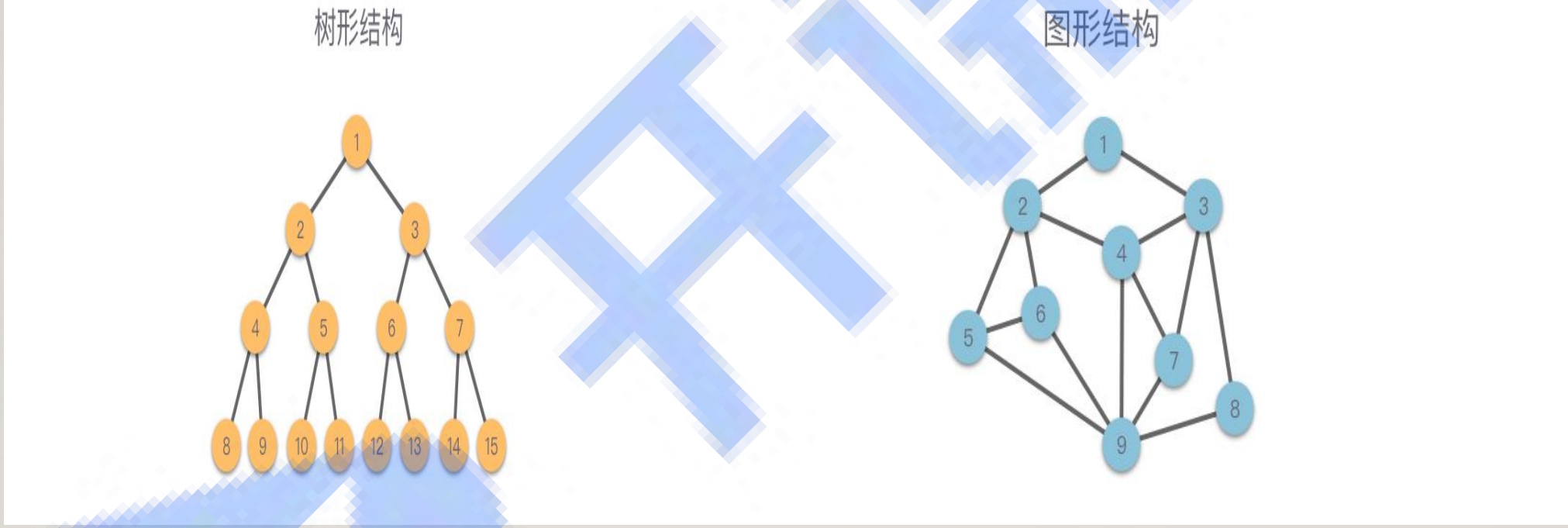
逻辑结构

- **集合结构**
- **线性结构**
- **树形结构**
- **图形结构**

逻辑结构

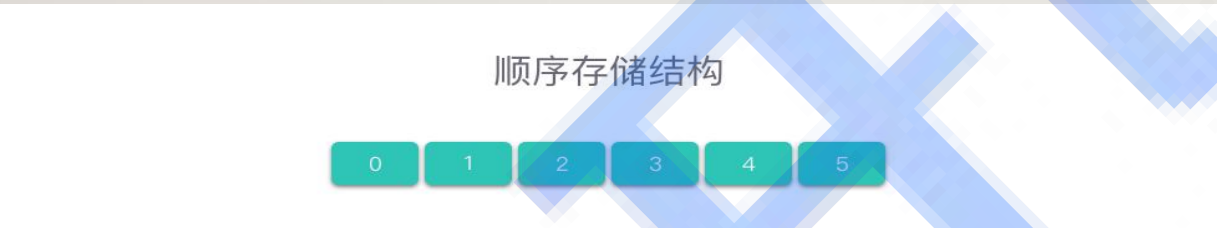


逻辑结构

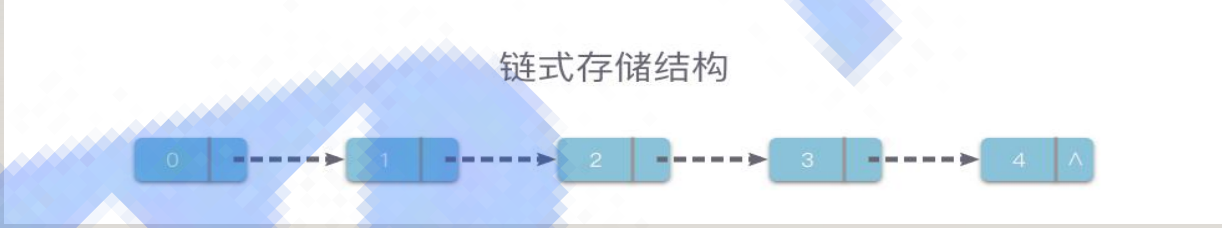


物理结构

- **顺序存储结构**



- **链式存储结构**



算法

输入
输出
有穷性
确定性
可行性

算法复杂度

算法复杂度（Algorithm complexity）：在问题的输入规模为 n 的条件下，程序的**时间**使用情况和**空间**使用情况。

时间复杂度

空间复杂度

算法复杂度

时间复杂度（Time Complexity）：在问题的输入规模为 n 的条件下，算法运行所需要花费的时间，可以记作为 O(n)。

1. 找到算法中的基本操作（基本语句）

2. 计算基本语句执行的次数的数量级

3. 用大O表示法表示时间复杂度

算法复杂度

```
int func(a,b){
    int sum = a+b;
}
```

```
int func(int n){
    int sum =0;
    for(int i=1;i<=n;i++){
        sum=+1;
    }
```

算法复杂度

```
int func(int n){
    int sum =0;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++)
        sum=+1;
}
```

算法复杂度

最佳时间复杂度：每个输入规模下用时最短的输入所对应的时间复杂度。

最坏时间复杂度：每个输入规模下用时最长的输入所对应的时间复杂度。

平均时间复杂度：每个输入规模下所有可能的输入所对应的平均用时复杂度（随机输入下期望用时的复杂度）。

算法复杂度

```
int find(int[] nums, int val){
        int pos =-1;
        for(int i=0;i<nums.length;i++){
                if(nums[i]==val){
                        pos= val;
                        break;
                }
        }
        return pos;
    }
```

算法复杂度

空间复杂度（Space Complexity）：在问题的输入规模为 n 的条件下，算法所占用的空间大小，可以记作为 S(n)。一般将 **算法的辅助空间** 作为衡量空间复杂度的标准。

算法复杂度

常见的时间复杂度有：O(1)、O(log2n)、O(n)、O(nlog2n)、O(n2)、O(n3)、O(2n)、O(n!)。

常见的空间复杂度有：O(1)、O(log2n)、O(n)、O(n2)。

数组

数组（Array）：一种线性表数据结构。它使用一组连续的内存空间，来存储一组具有相同类型的数据。

数组

C/C++

```
int arr[3][4] = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};
```

java

```
int[][] arr = new int[3][]{ {1,2,3}, {4,5}, {6,7,8,9}};
```

数组

访问：a[i]，　a[i][j]

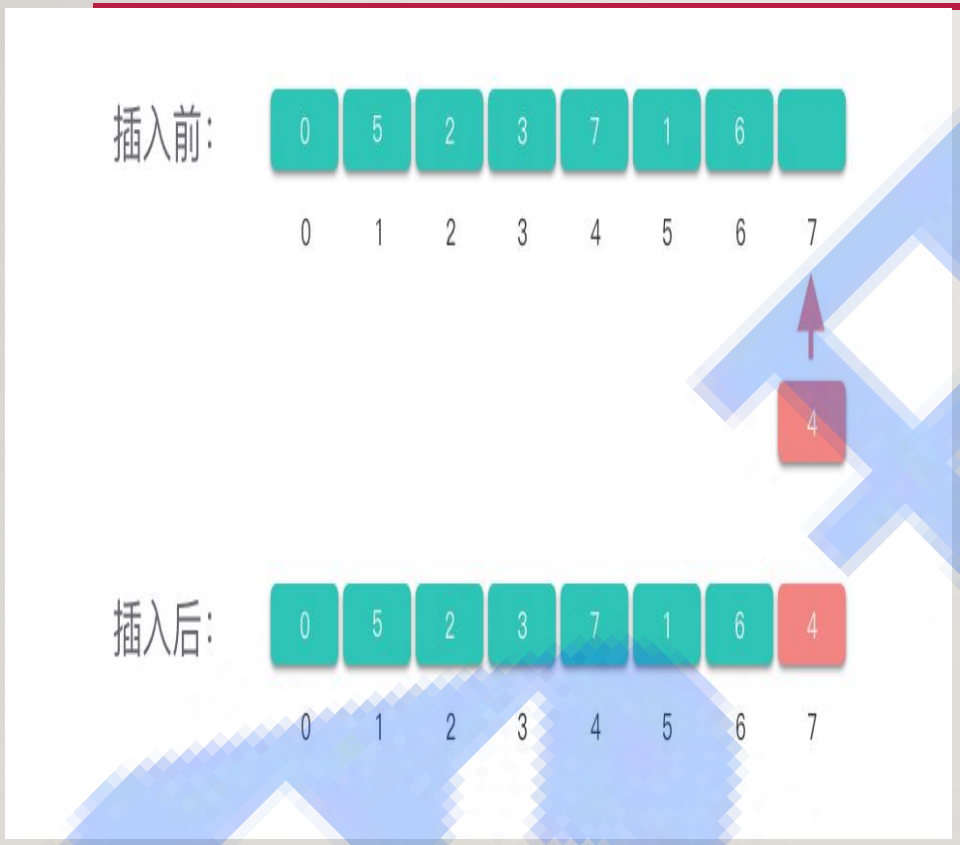
查找：**查找数组中元素值为 val 的位置**

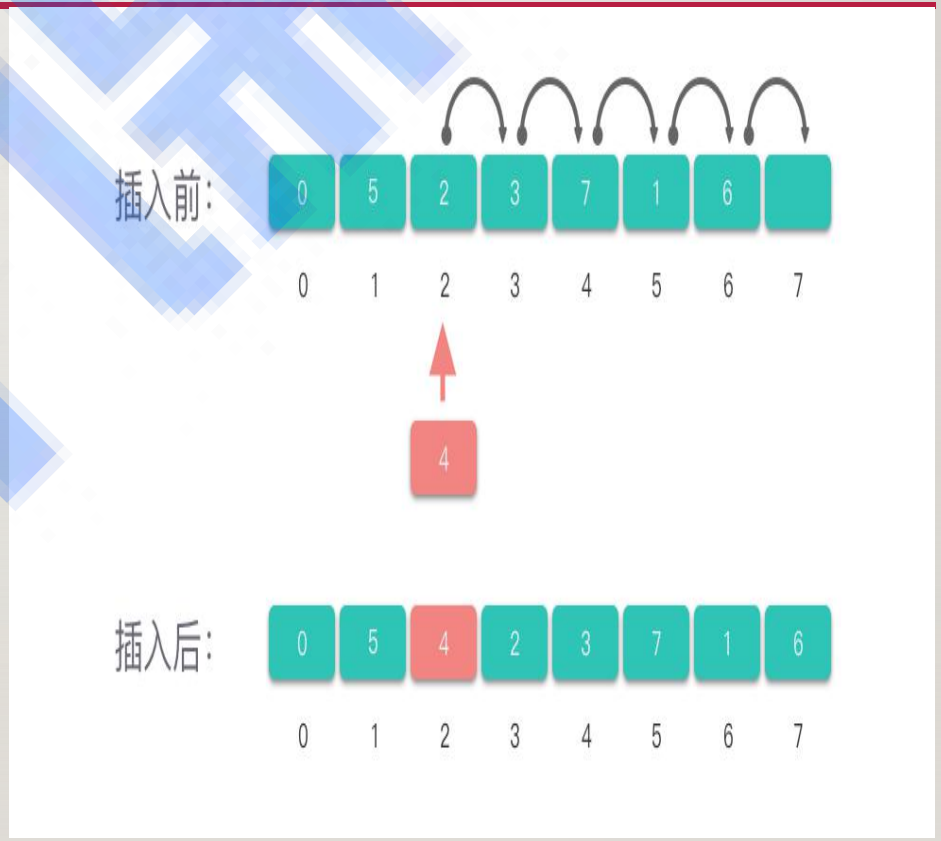
插入

改变

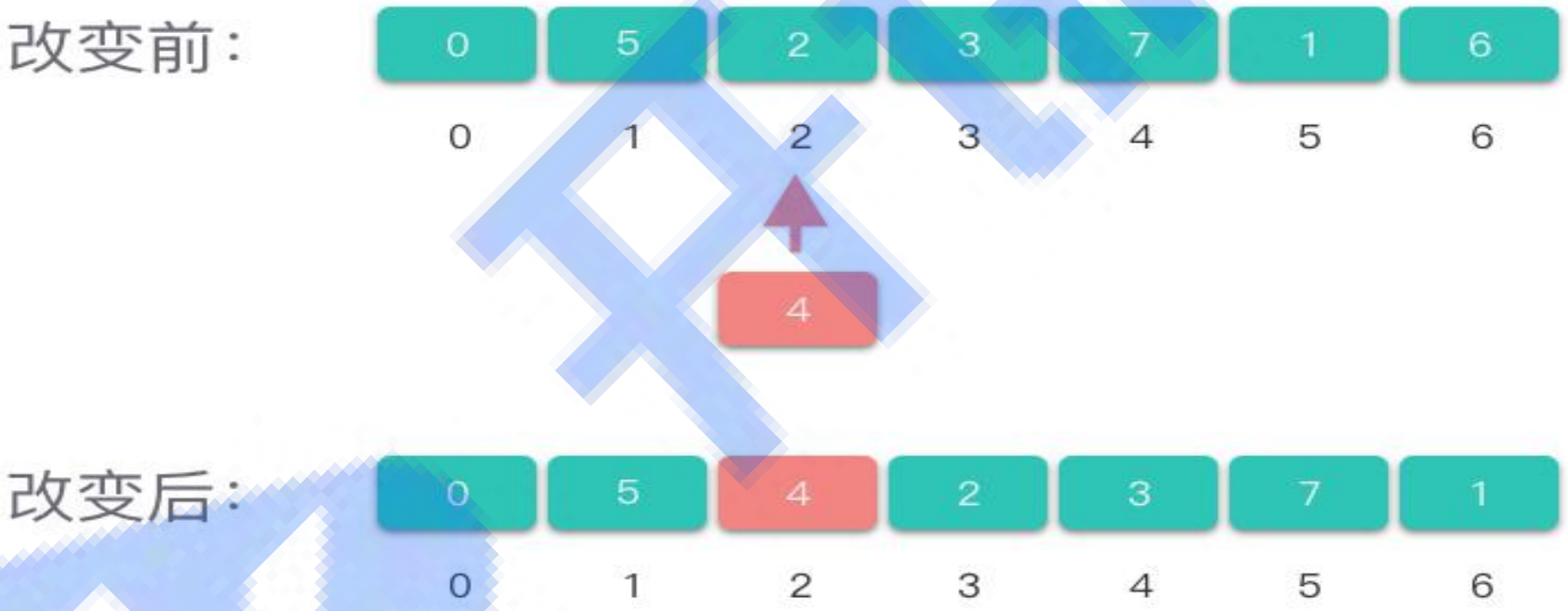
删除

数组

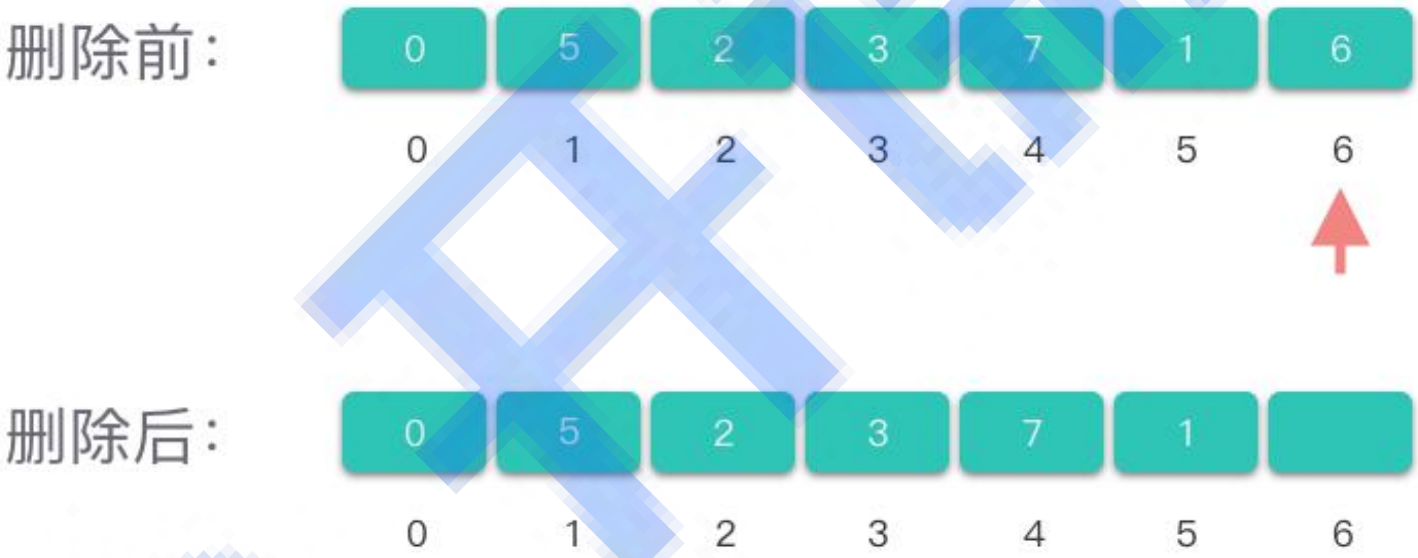

插入前:
0 5 2 3 7 1 6
0 1 2 3 4 5 6 7
4
插入后:
0 5 2 3 7 1 6 4
0 1 2 3 4 5 6 7


插入前:
0 5 2 3 7 1 6
0 1 2 3 4 5 6 7
4
插入后:
0 5 4 2 3 7 1 6
0 1 2 3 4 5 6 7

数组



数组



数组

给定一个由 整数 组成的 非空 数组所表示的非负整数，在该数的基础上加一。最高位数字存放在数组的首位， 数组中每个元素只存储单个数字。你可以假设除了整数 0 之外，这个整数不会以零开头。

示例 1:
输入: digits = [1,2,3]
输出: [1,2,4]
解释: 输入数组表示数字 123。

示例 2:
输入: digits = [4,3,2,1] 输出: [4,3,2,2]
解释: 输入数组表示数字 4321。

数组

```
class Solution {
    public int[] plusOne(int[] digits) {
        int n = digits.length;
        for (int i = n - 1; i >= 0; --i) {
            if (digits[i] != 9) {
                ++digits[i];
                for (int j = i + 1; j < n; ++j) {
                    digits[j] = 0;
                }
                return digits;
            }
        }

        // digits 中所有的元素均为 9
        int[] ans = new int[n + 1];
        ans[0] = 1;
        return ans;
    }
}
```

字符串

字符串（String）：简称为串，是由零个或多个字符组成的有限序列 str = "Hello World"

字符串名称:
字符串的值:
字符变量:
字符串的长度:
空串：零个字符构成的串也成为**「空字符串（Null String）」**，它的长度为 0，可以表示为 ""。
子串：字符串中任意个连续的字符组成的子序列称为该字符串的**「子串（Substring）」**。并且有两种特殊子串，起始于位置为 0、长度为 k 的子串称为**「前缀（Prefix）」**。而终止于位置 n - l、长度为 k 的子串称为**「后缀（Suffix）」**。
主串：包含子串的字符串相应的称为**「主串」**。

字符串

字符串

字符串匹配问题;

子串相关问题;

前缀 / 后缀相关问题;

回文串相关问题;

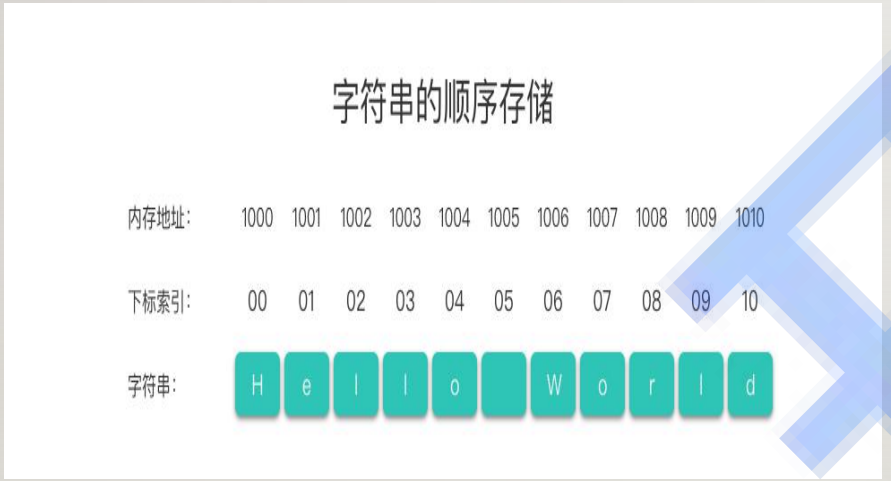
子序列相关问题。

字符串

字符串比较

如果说两个字符串 str1 和 str2 相等，则必须满足两个条件：
字符串 str1 和字符串 str2 的长度相等。
字符串 str1 和字符串 str2 对应位置上的各个字符都相同。

字符串的存储结构


字符串的顺序存储
内存地址: 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010
下标索引: 00 01 02 03 04 05 06 07 08 09 10
字符串: H e l l o W o r l d


字符串的链式存储
内存地址: 1000 2016 5022
链表节点: H e l l o W o r l d # ^

字符串

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 s 的形式给出。

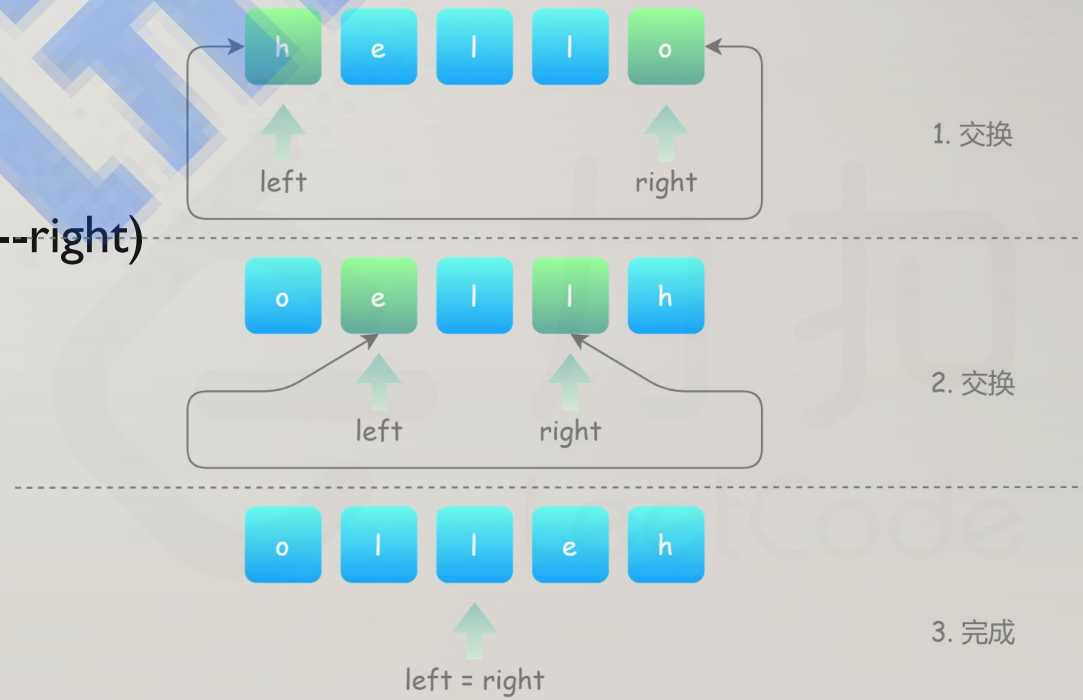
不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

示例 1:
输入：s = ["h","e","l","l","o"]
输出：["o","l","l","e","h"]

示例 2:
输入：s = ["H","a","n","n","a","h"]
输出：["h","a","n","n","a","H"]

字符串

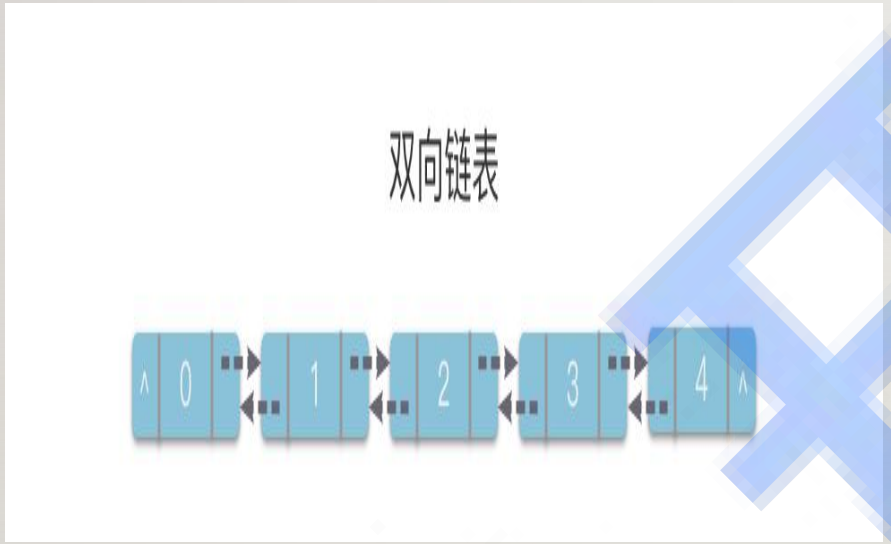
```
class Solution {
    public void reverseString(char[] s) {
        int n = s.length;
        for (int left = 0, right = n - 1; left < right; ++left, --right)
{
            char tmp = s[left];
            s[left] = s[right];
            s[right] = tmp;
        }
    }
}
```

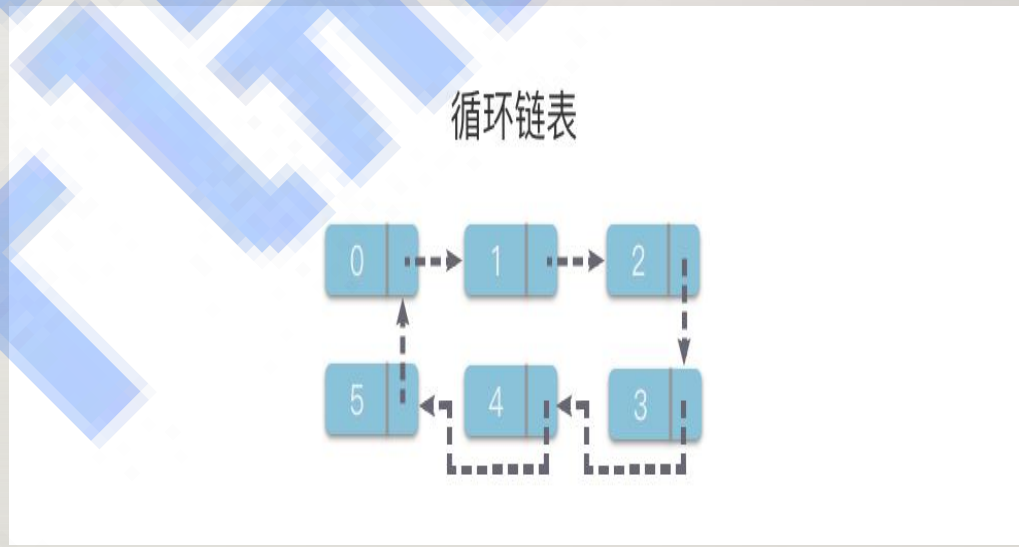


链表

链表（Linked List）：一种线性表数据结构。它使用一组任意的存储单元（可以是连续的，也可以是不连续的），来存储一组具有相同类型的数据。

链表





链表

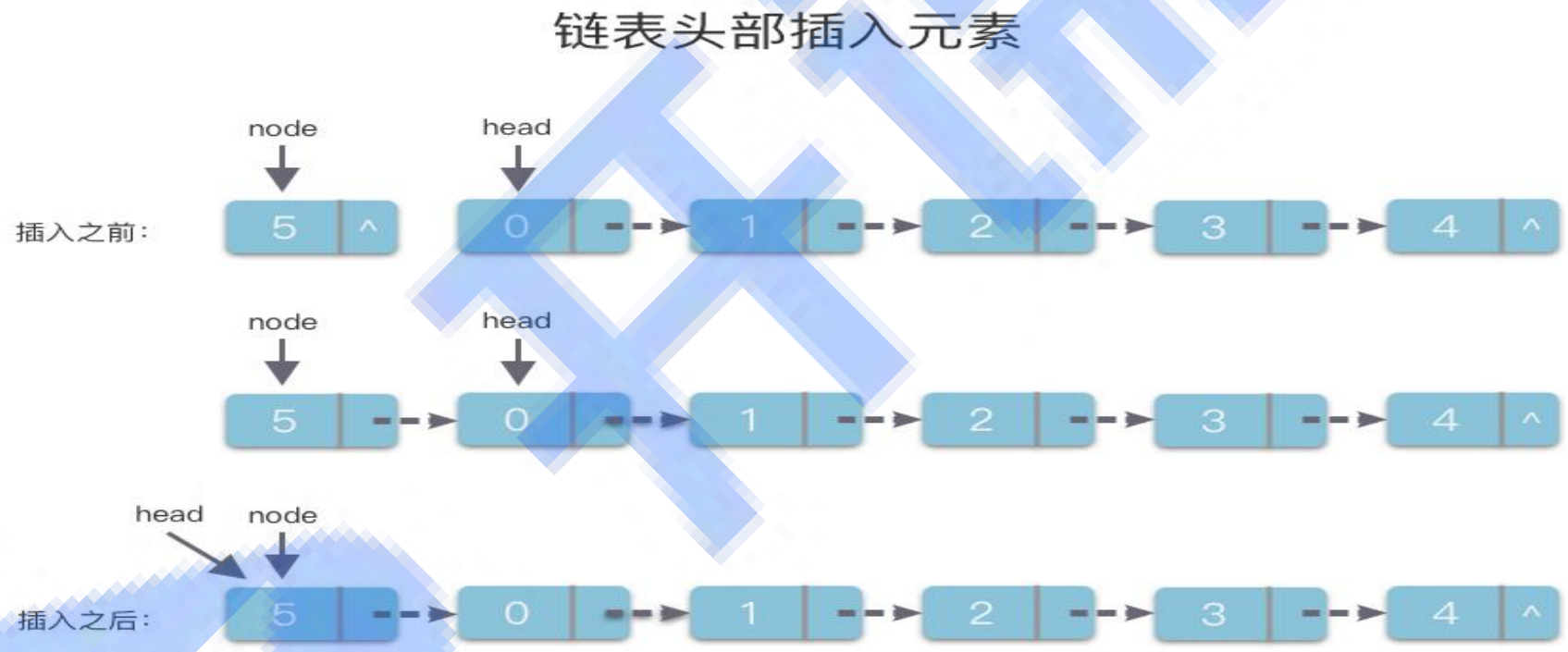
增，删，改，查

```
private class ListNode{

private ListNode next;
private mytype data;

public ListNode(mytype data){
   this.data=data;
 }
```

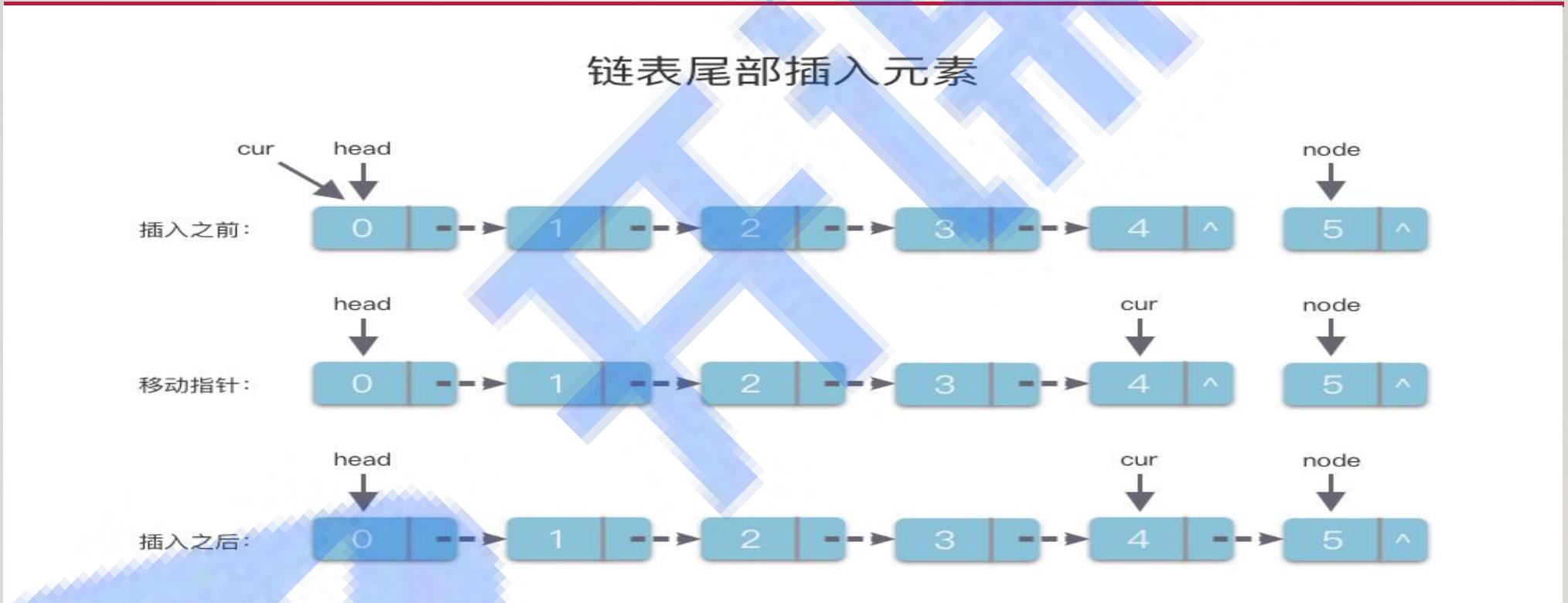
链表

```
public void add(int newval) {
        ListNode newNode = new
ListNode(newval);
        if(this.next == null)
            this.next = newNode;
        else
            this.next.add(newval);
    }
```

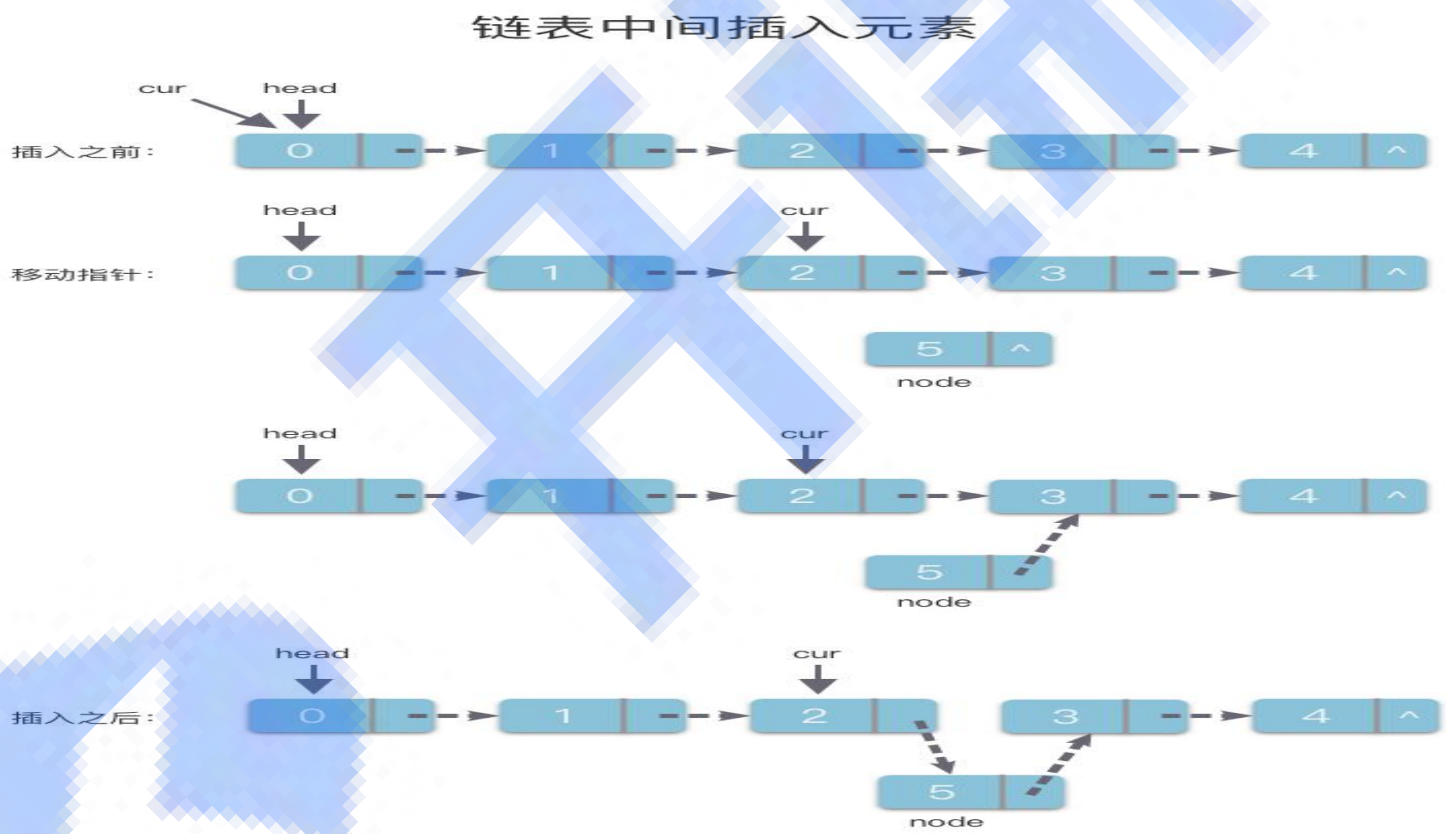
链表


链表头部插入元素
node
head
插入之前：
5
^
0
1
2
3
4
^
node
head
5
0
1
2
3
4
^
head
node
插入之后：
5
0
1
2
3
4
^

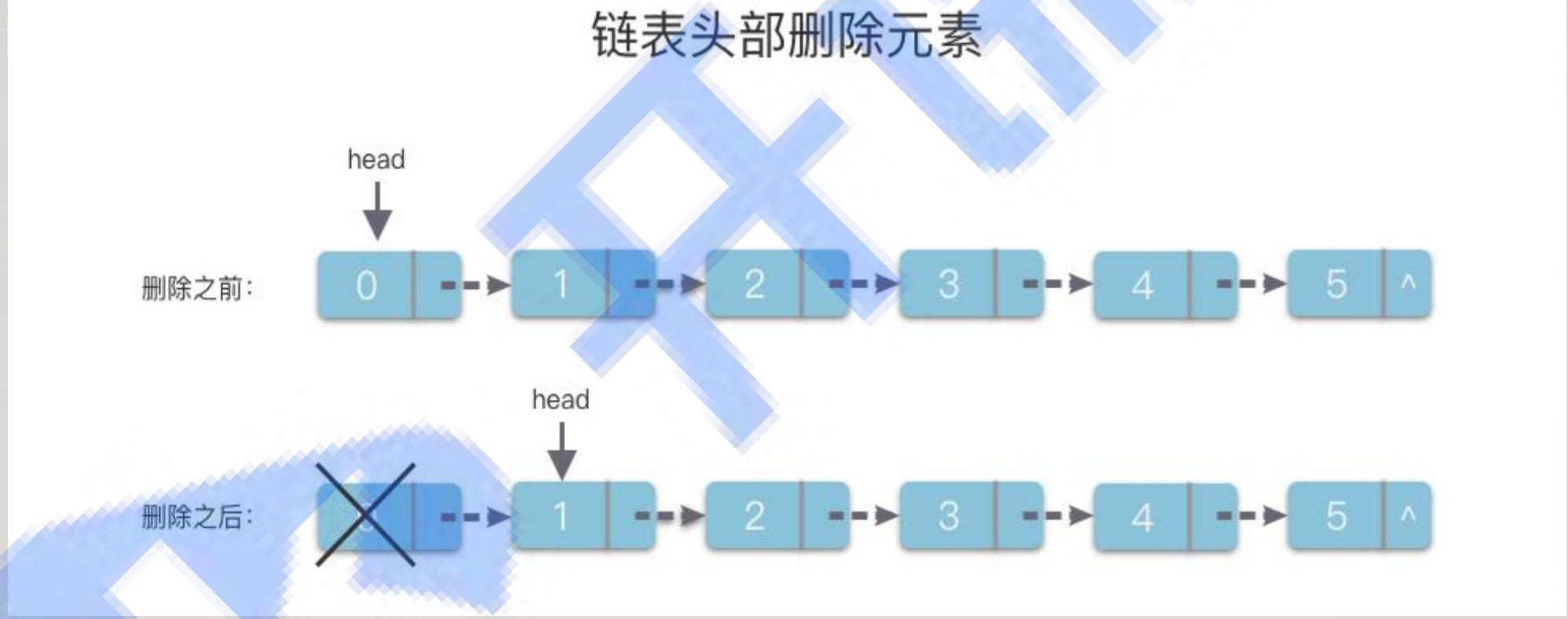
链表


链表尾部插入元素
cur
head
node
插入之前：
0
1
2
3
4
^
5
^
head
cur
node
移动指针：
0
1
2
3
4
^
5
^
head
cur
node
插入之后：
0
1
2
3
4
5
^

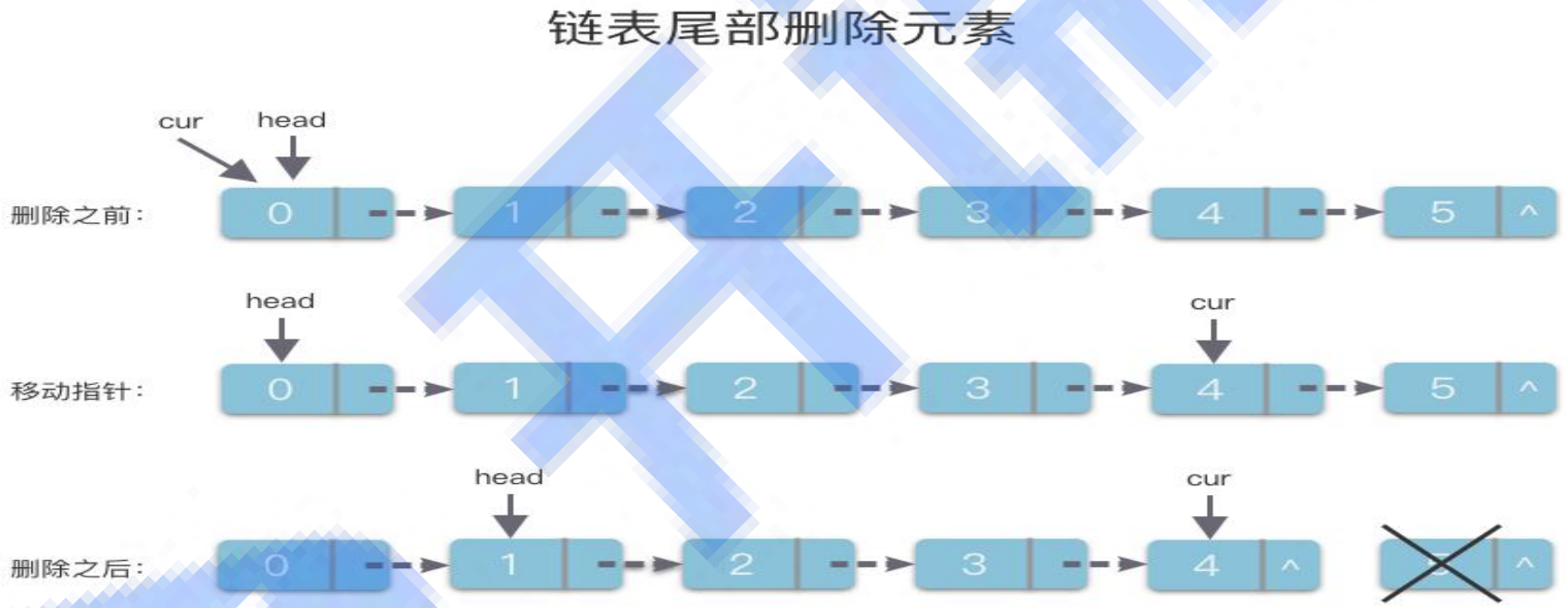
链表



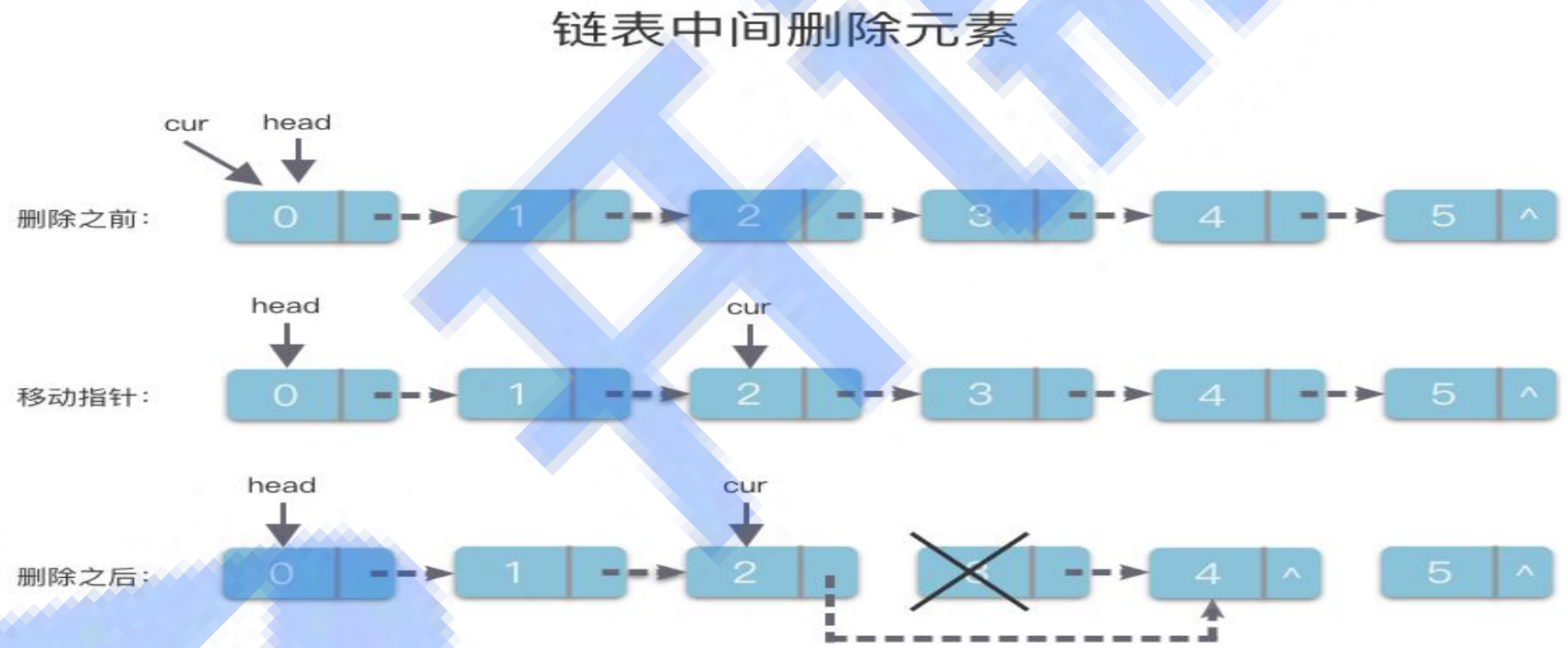
链表



链表


链表尾部删除元素
cur
head
删除之前：
0
1
2
3
4
5
^
head
cur
移动指针：
0
1
2
3
4
5
^
head
cur
删除之后：
0
1
2
3
4
^
^

链表

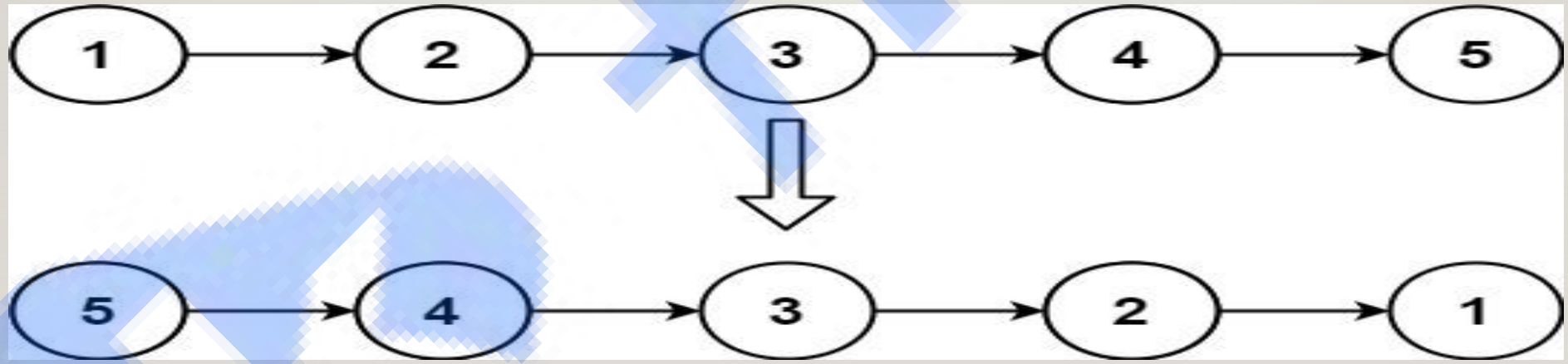

链表中间删除元素
cur
head
删除之前：
0
1
2
3
4
5
^
head
cur
移动指针：
0
1
2
3
4
5
^
head
cur
删除之后：
0
1
2
4
5
^

链表

给你单链表的头节点 head ， 请你反转链表， 并返回反转后的链表。

示例 1：

输入： head = [1,2,3,4,5]
输出： [5,4,3,2,1]

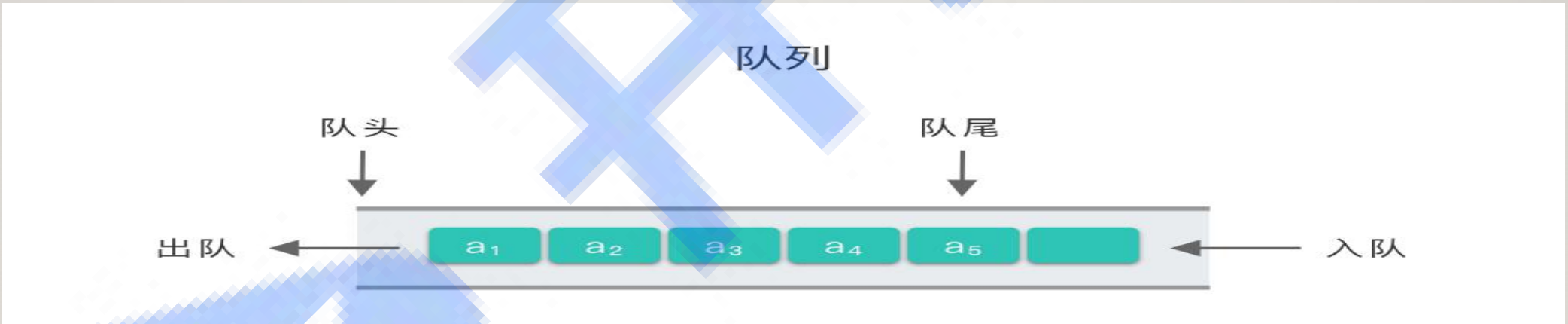


链表

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

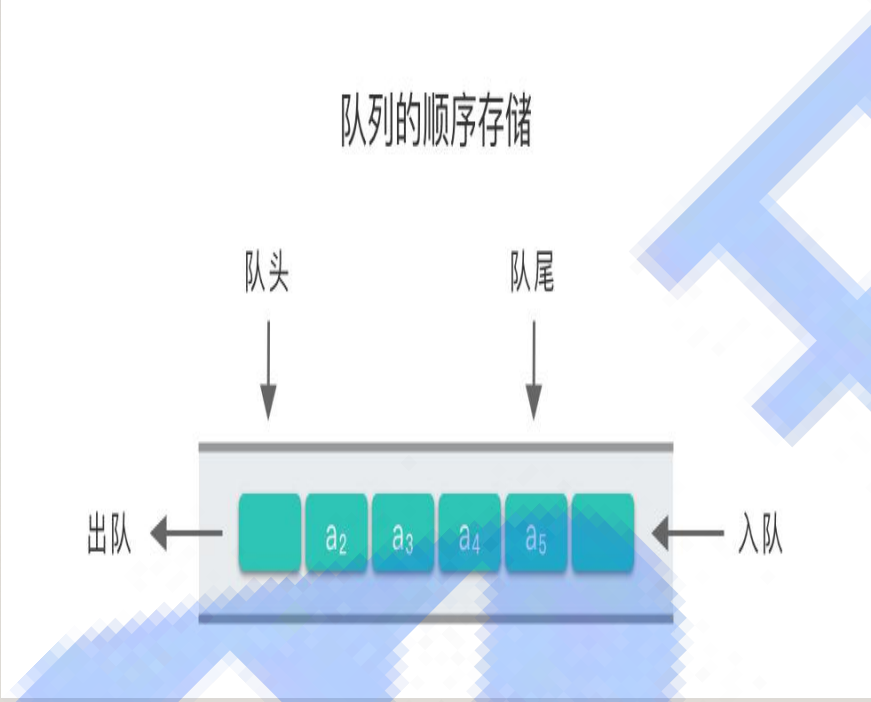
队列

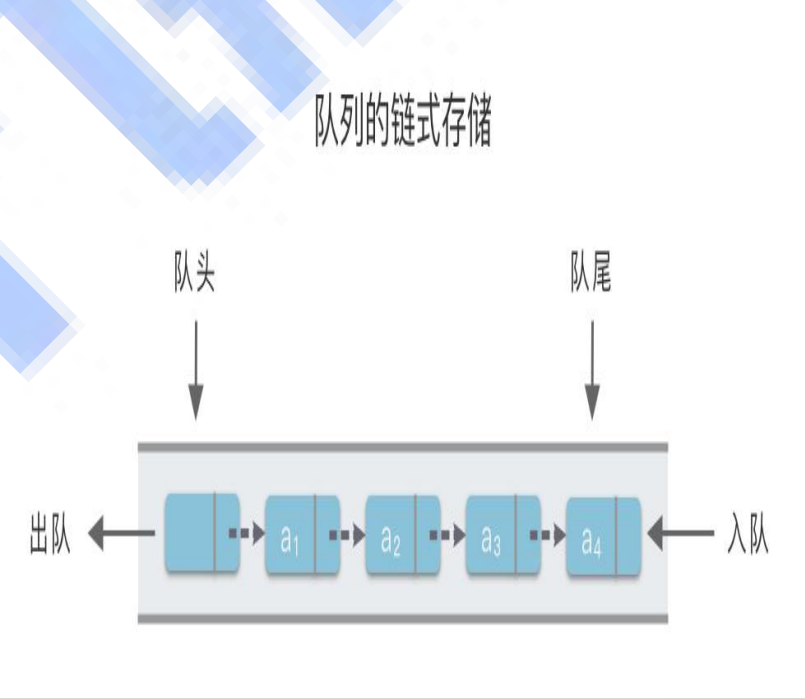
队列（Queue）：简称为队，一种线性表数据结构，是一种只允许在表的一端进行插入操作，而在表的另一端进行删除操作的线性表。



队列

队列有两种存储表示方法：「**顺序存储的队列**」和「**链式存储的队列**」





队列

队列有两种基本操作：**「插入操作」**和**「删除操作」**。
队列的插入操作又称为「入队」。
队列的删除操作又称为「出队」

队列

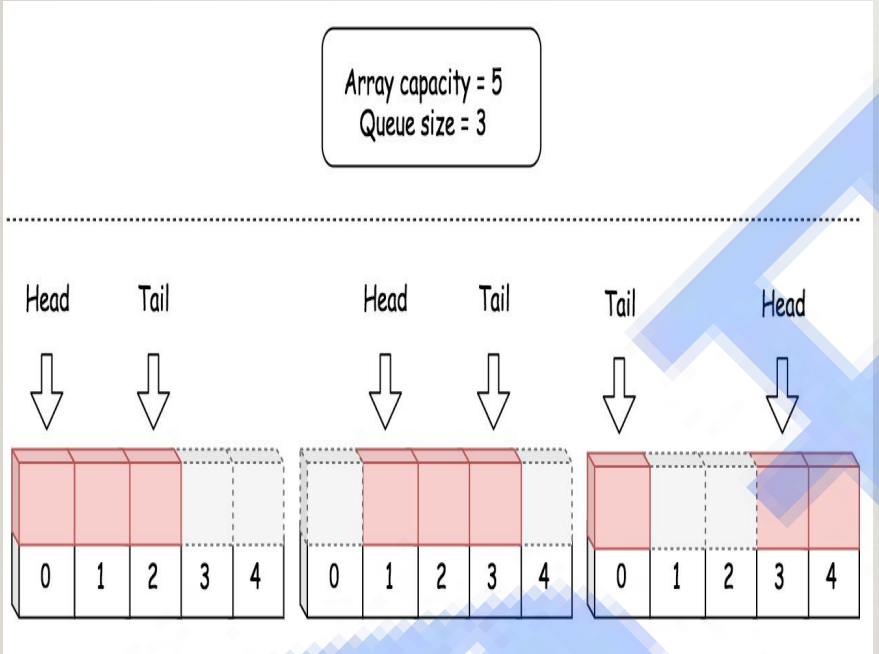
设计你的循环队列实现。 循环队列是一种线性数据结构，其操作表现基于 FIFO（先进先出）原则并且队尾被连接在队首之后以形成一个循环。它也被称为“环形缓冲器”。

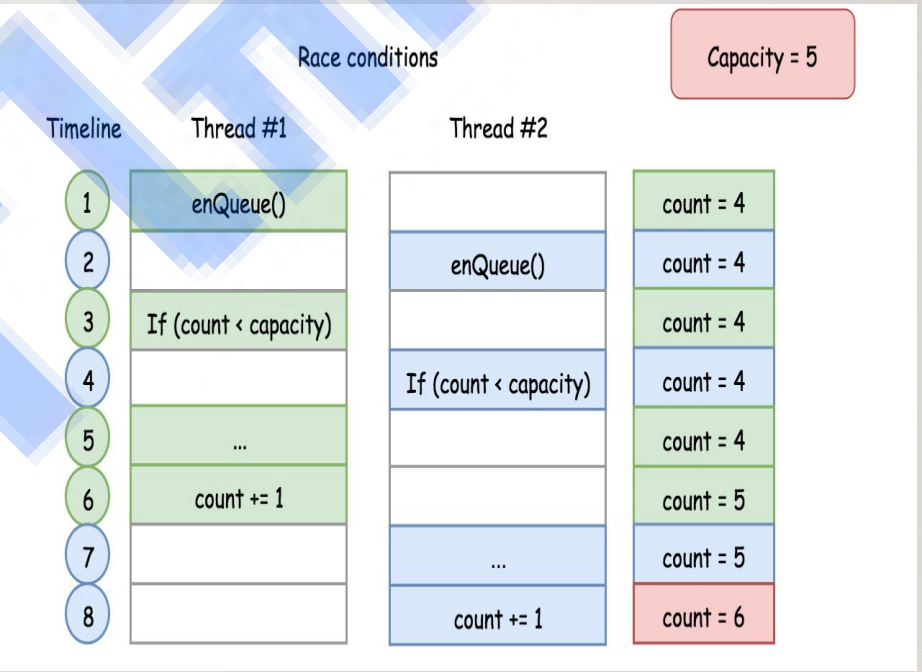
循环队列的一个好处是我们可以利用这个队列之前用过的空间。在一个普通队列里，一旦一个队列满了，我们就不能插入下一个元素，即使在队列前面仍有空间。但是使用循环队列，我们能使用这些空间去存储新的值。

你的实现应该支持如下操作：

MyCircularQueue(k): 构造器，设置队列长度为 k 。
Front: 从队首获取元素。如果队列为空，返回 -1 。
Rear: 获取队尾元素。如果队列为空，返回 -1 。
enQueue(value): 向循环队列插入一个元素。如果成功插入则返回真。
deQueue(): 从循环队列中删除一个元素。如果成功删除则返回真。
isEmpty(): 检查循环队列是否为空。
isFull(): 检查循环队列是否已满。

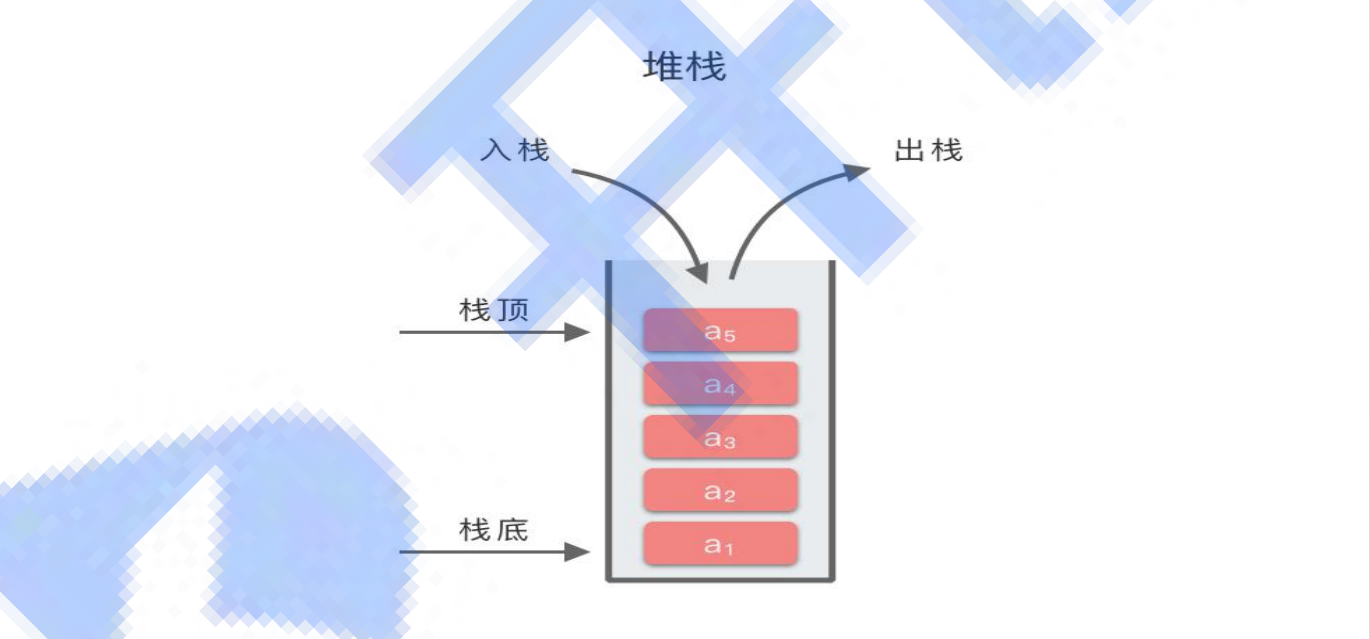
队列


Array capacity = 5
Queue size = 3
Head
Tail
Head
Tail
Tail
Head
0
1
2
3
4
0
1
2
3
4
0
1
2
3
4


Race conditions
Capacity = 5
Timeline
Thread #1
Thread #2
1
2
3
4
5
6
7
8
enQueue()
If (count < capacity)
...
count += 1
enQueue()
If (count < capacity)
...
count += 1
count = 4
count = 4
count = 4
count = 4
count = 4
count = 5
count = 5
count = 6

堆栈

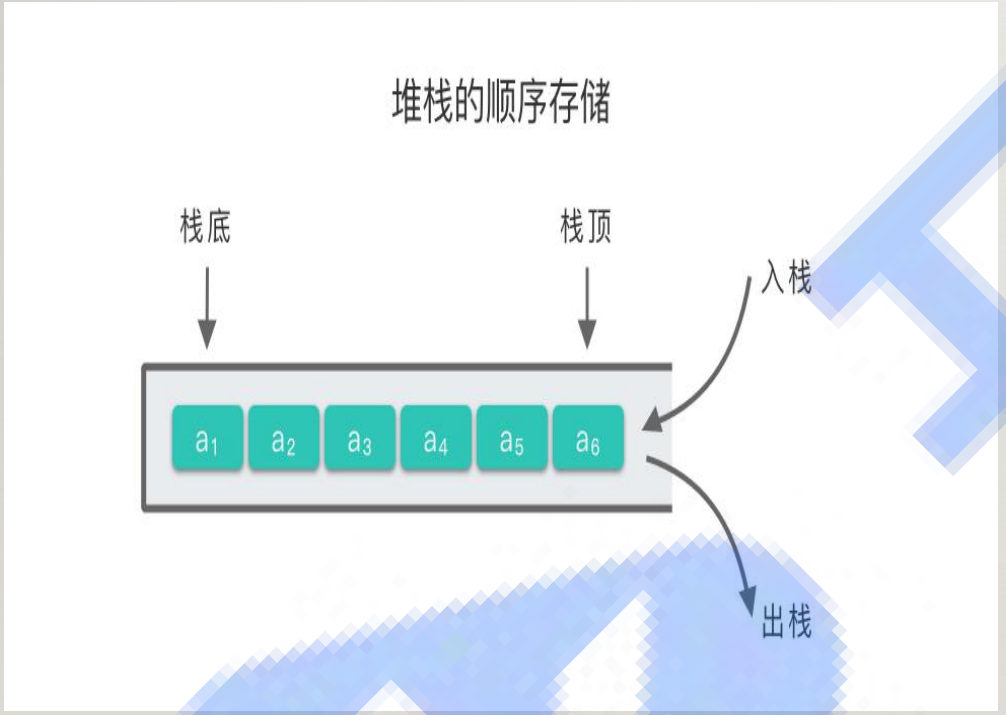
堆栈（Stack）：简称为栈。一种线性表数据结构，是一种只允许在表的一端进行插入和删除操作的线性表。

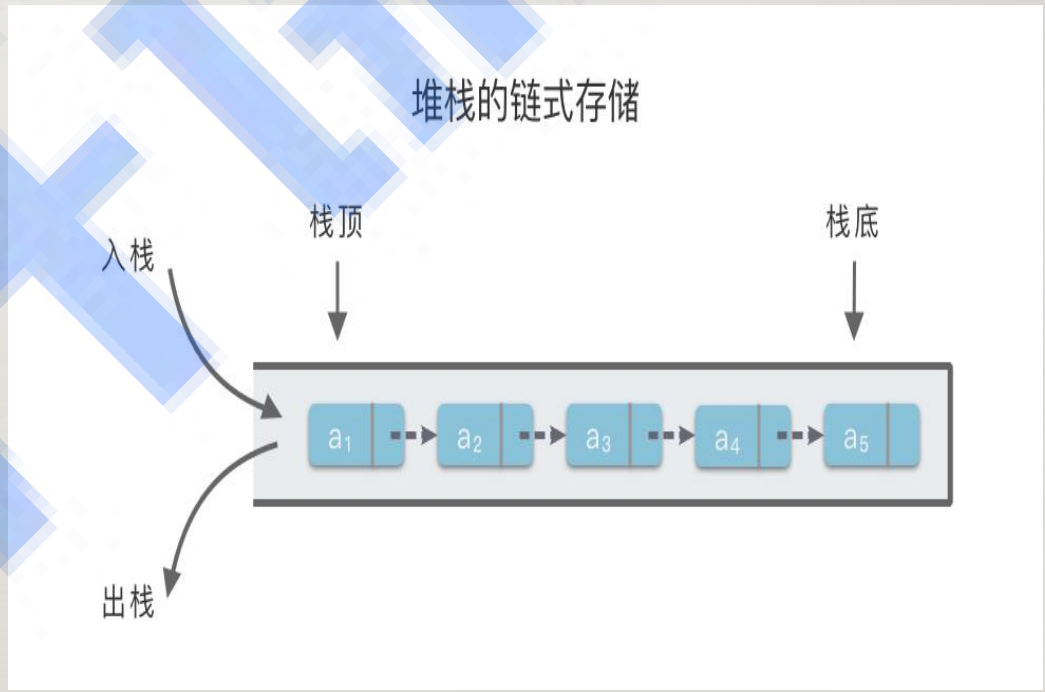


堆栈

初始化空栈：创建一个空栈，定义栈的大小 size，以及栈顶元素指针 top。
判断栈是否为空：当堆栈为空时，返回 True。当堆栈不为空时，返回 False。一般只用于栈中删除操作和获取当前栈顶元素操作中。
判断栈是否已满：当堆栈已满时，返回 True，当堆栈未满时，返回 False。一般只用于顺序栈中插入元素和获取当前栈顶元素操作中。
插入元素（进栈、入栈）：相当于在线性表最后元素后面插入一个新的数据元素。并改变栈顶指针 top 的指向位置。
删除元素（出栈、退栈）：相当于在线性表最后元素后面删除最后一个数据元素。并改变栈顶指针 top 的指向位置。
获取栈顶元素：相当于获取线性表中最后一个数据元素。与插入元素、删除元素不同的是，该操作并不改变栈顶指针 top 的指向位置。

堆栈


堆栈的顺序存储
栈底
栈顶
入栈
出栈
a1
a2
a3
a4
a5
a6


堆栈的链式存储
栈顶
栈底
入栈
出栈
a1
a2
a3
a4
a5

堆栈

操作系统中的函数调用栈，浏览器中的前进、后退功能

翻转一组元素的顺序、铁路列车车辆调度。

堆栈

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ， 判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。
左括号必须以正确的顺序闭合。

示例 1：

输入：s = "()"
输出：true

堆栈

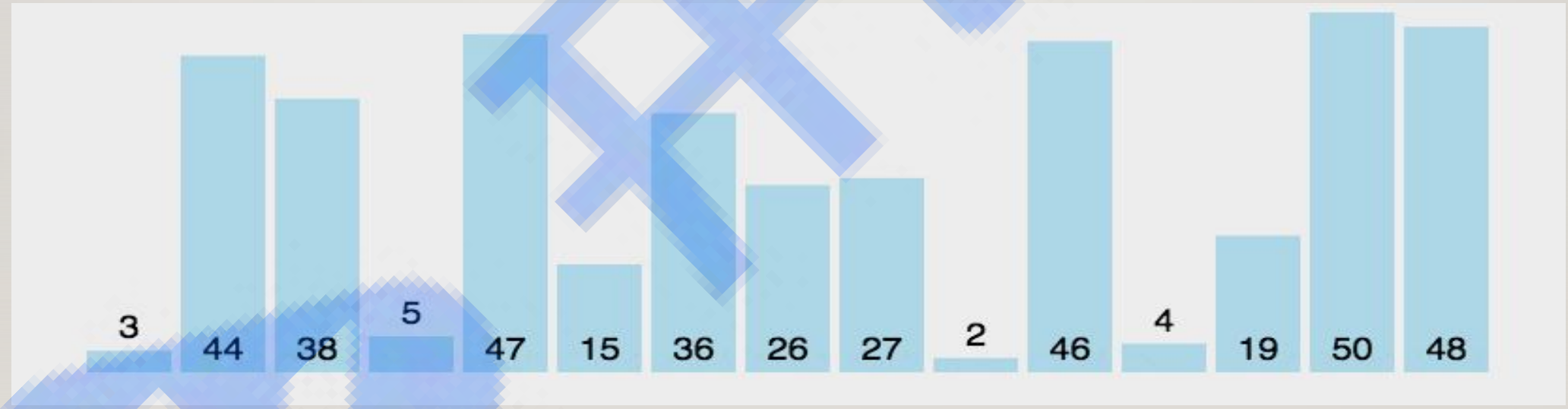
```
class Solution {
    public boolean isValid(String s) {
        int n = s.length();
        if (n % 2 == 1) {
            return false;
        }

        Map<Character, Character> pairs = new HashMap<Character, Character>() {{
            put(')', '(');
            put(']', '[');
            put('}', '{');
        }};
        Deque<Character> stack = new LinkedList<Character>();
        for (int i = 0; i < n; i++) {
            char ch = s.charAt(i);
            if (pairs.containsKey(ch)) {
                if (stack.isEmpty() || stack.peek() != pairs.get(ch)) {
                    return false;
                }
                stack.pop();
            } else {
                stack.push(ch);
            }
        }
        return stack.isEmpty();
    }
}
```

数组的排序

冒泡
选择
插入
快速

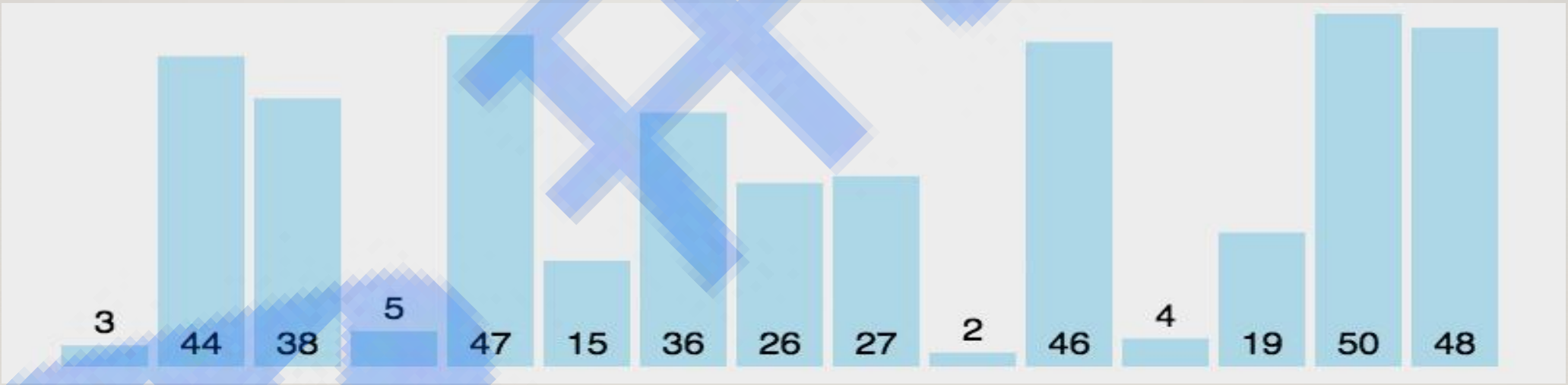
冒泡排序

思想：第 i (i = 1，2，…) 趟排序时从序列中前 n - i + 1 个元素的第 1 个元素开始，相邻两个元素进行比较，若前者大于后者，两者交换位置，否则不交换。



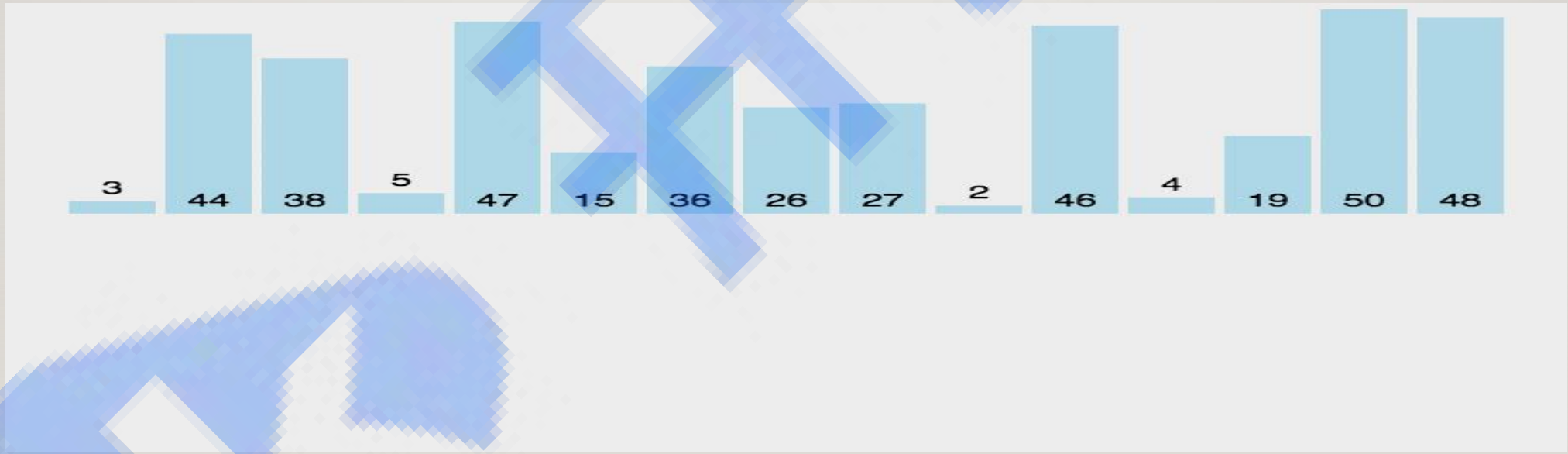
选择排序

第 i 趟排序从序列的后 n − i + 1 (i = 1, 2, …, n − 1) 个元素中选择一个值最小的元素与该 n - i + 1 个元素的最前面那个元素交换位置，即与整个序列的第 i 个位置上的元素交换位置。如此下去，直到 i == n − 1，排序结束。



插入排序

将整个序列切分为两部分：前 i - l 个元素是有序序列，后 n - i + l 个元素是无序序列。每一次排序，将无序序列的首元素，在有序序列中找到相应的位置并插入。



快速排序

通过一趟排序将无序序列分为独立的两个序列，第一个序列的值均比第二个序列的值小。然后递归地排列两个子序列，以达到整个序列有序。