# 1.评判一个算法的好坏？

## 大O表示法（Big O）

它表示的是数据规模为 n时所 对应的复杂度。

**原则：忽略常数、系数、阶数：**

|  |
| --- |
| 9 >> O(1)  2n + 3 >> O(n)  n2 + 2n + 6 >> O(n2)  4n3 + 3n2 + 22n + 100 >> O(n3)  对数阶一般省略底数，对数阶乘以一个常数，可以替换底数。  log29 ∗ log9n= log2n  所以 O(log2n) 、O(log9n) 统称为 O(logn) |

## 常见的复杂度





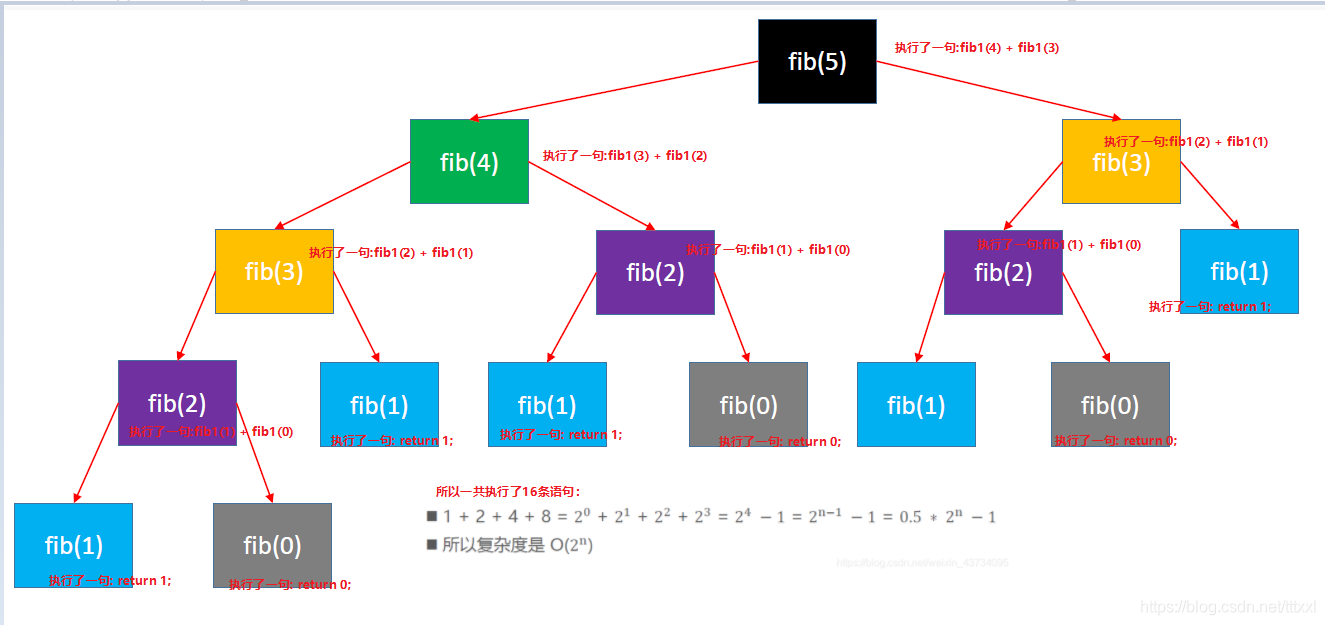
## 多个数据规模的情况

时间复杂度：O(n + k)

|  |
| --- |
| public static void test(int n, int k){  for(int i = 0; i < n; i++){  System.*out*.println("test");  }  for (int i = 0; i < k; i++){  System.*out*.println("test");  } } |

## 递归情况复杂度分析

|  |
| --- |
| public static int fib1(int n) {  if (n <= 1) return n;  return *fib1*(n - 1) + *fib1*(n - 2); } |

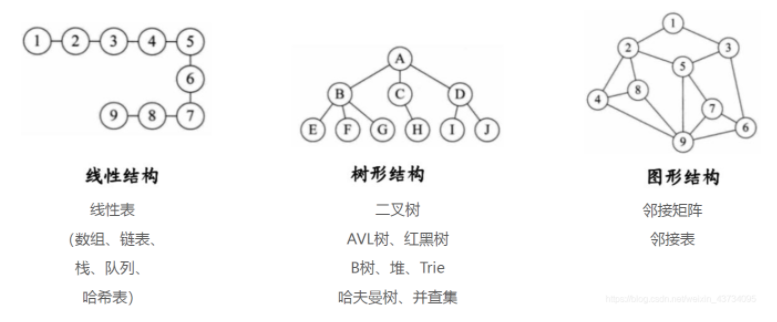


## 算法的优化方向

* 用尽量少的存储空间
* 用尽量少的执行步骤（执行时间）
* 根据情况，可以
  + 空间换时间
  + 时间换空间

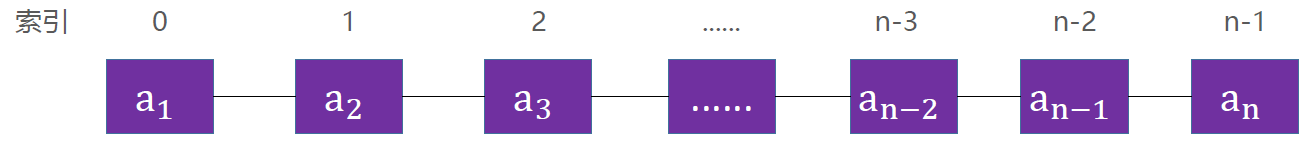
# 2.什么是数据结构？

数据结构是计算机存储、组织数据的方式；



在实际应用中，根据使用场景来选择最合适的数据结构；

## 1.线性表

* 线性表是具有 n 个相同类型元素的有限序列（ n ≥ 0 ）；
* **在逻辑上具有索引和数据两个部分**：是否能通过索引能直接访问到数据部分，要看具体的线性表类型（典型代表：数组和链表）。  
  

a1 是首节点（首元素）， an 是尾结点（尾元素）

a1 是 a2 的前驱， a2 是 a1 的后继

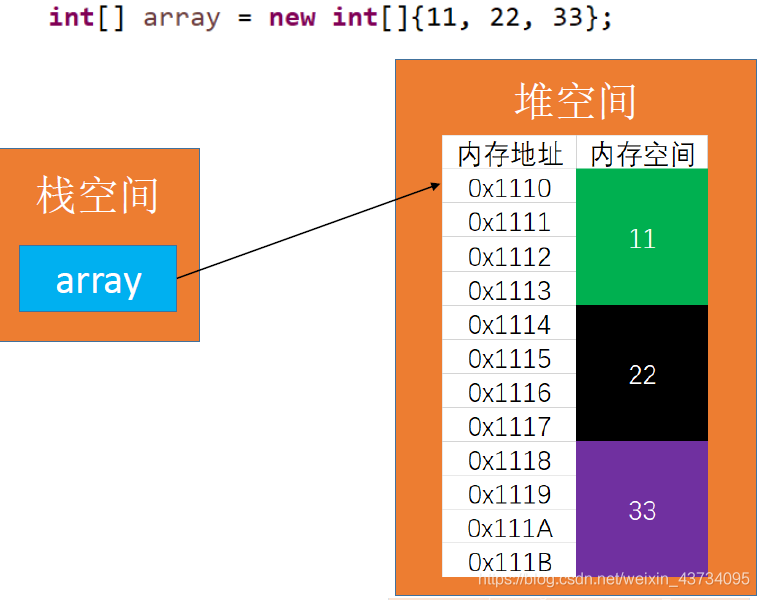
常见的线性表有：

* 数组
* 链表
* 栈
* 队列
* 哈希表（散列表）

# 线性表

## 数组

数组是一种顺序存储的线性表，所有元素的****内存地址是连续****的；



数组都有个致命的缺点 ：

数组array初始化后，其容量大小就是固定好了，以后无法动态修改数组容量。数组元素存满后，就无法再存放数据了，因为不能扩大数组的大小了。

## 动态数组（Dynamic Array）

实际开发中，我们更希望数组的容量是可以动态改变的，所以我们会自己实现一个可以动态扩容的数组。那么我们需要考虑一个问题：我们自己实现的可扩容的数组需要提供哪些API接口给别人调用？

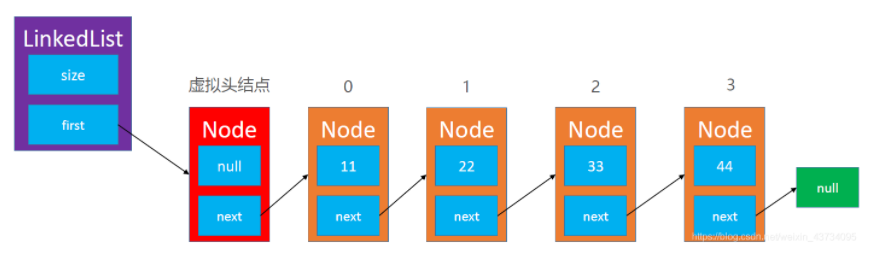
### 2.1动态数组接口设计

|  |
| --- |
| int size(); *// 元素的数量* boolean isEmpty(); *// 是否为空* boolean contains (E element); *// 是否包含某个元素* void add (E element); *// 添加元素到最后面* E get (int index); *// 返回index位置对应的元素* E set (int index, E element); *// 设置index位置的元素* void add (int index, E element); *// 往index位置添加元素* E remove (int index); *// 删除index位置对应的元素* int indexOf (E element); *// 查看元素的位置* void clear (); *// 清除所有元素* |

## 3.链表

### 带虚拟头结点的单向链表

有时候为了让代码更加精简，统一所有节点的处理逻辑，可以在最前面增加一个虚拟的头结点（不存储数据）



之前需要单独处理头结点。

现在处理头结点，first节点就等价于node(-1)

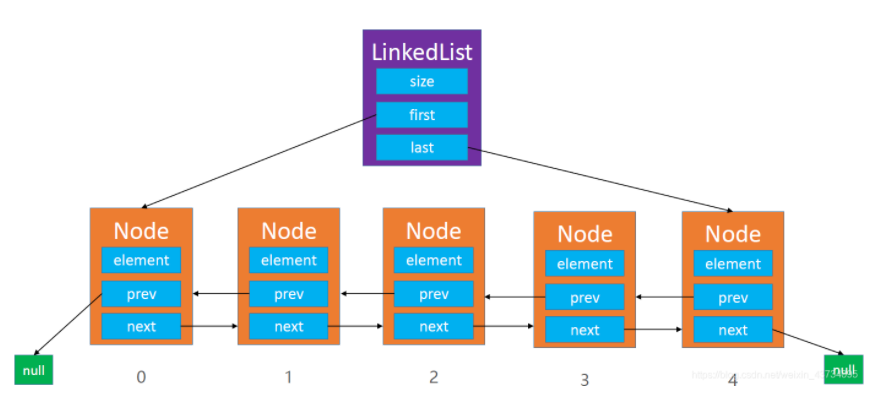
### 动态数组、链表复杂度分析



以上为根据索引来进行增删改查

### 双向链表

* **结构：**



* **原型：**

|  |
| --- |
| public class DoubleLinkedList<E> extends AbstractList<E>{    Node<E> first;  Node<E> last;    private class Node<E>{  E element;  Node<E> prev;  Node<E> next;    public Node(){}  public Node(E element,Node<E> prev,Node<E> next){  this.element = element;  this.prev = prev;  this.next = next;  }  }   } |