参考：<https://blog.csdn.net/weixin_38483589/article/details/84262167>

对于下面一段代码，分析其时间复杂度时，需要结合具体的情况来判断。

public int find(int[] arr, int target) {

int n = arr.length;

for (int i = 0; i < n; i++) {

// 依次遍历数组，如果找到和目标元素相同的值，在返回该值所在下标

if (arr[i] == target) {

return i;

}

}

return -1;

}

该程序是在给定的数组中查找指定的元素。

时间复杂度的分析如下：

最好时间复杂度：如果target正好在数组的第一个位置，则时间复杂度为

最坏时间复杂度：target在数组的最后一个位置，则时间复杂度为。

平均时间复杂度：由于target在数组中的位置不同，导致时间复杂度呈现量级差异。target在数组中，位置有n种情况，加上不在数组，总共有n+1种情况。每种情况需要访问的元素次数依次为：1,2,3,…n,n，平均访问元素次数为：



则平均时间复杂度为：

均摊时间复杂度：下段代码模拟动态数组扩容

public class IntArray {

// 记录数组中已有元素个数

int count = 0;

// 声明数组

int[] arr;

public IntArray(int n){

// 初始化数组，这里只是举例说明，不考虑n<0等异常情况

arr = new int[n];

}

/\*\*

\* 插入元素

\* @param value

\*/

public void insert(int value) {

// 数组已经存满，进行扩容操作，然后将之前的元素拷贝到新数组中

if (count >= arr.length) {

// 新建一个大小为之前数组2倍的新数组

int[] arr2 = new int[2\*arr.length];

// 将之前数组中元素copy到新数组中

for (int i = 0;i<arr.length;i++) {

arr2[i] = arr[i];

}

// 将新数组赋值给原数组（扩容后的数组代替原来的数组）

arr = arr2;

}

// 数组没满，直接将值插入到数组中即可

arr[count] = value;

count++;

}

}

假如数组元素已满，插入新数据时新建一个长度为原数组2倍的数组，然后把原数组中的值拷贝到新数组中，将新数组替换为原数组，并插入新数据。

当数组未满时，插入数据的时间复杂度为，当数组已满时，插入数据的时间复杂度为。每经历n次时间复杂度为的操作，便经历一次时间复杂度为的操作。出现低级别时间复杂度的次数多，出现高级别时间复杂度的次数少，均摊时间复杂度为。

总结：

1. 代码在不同情况下复杂度出现量级差别，则用平均时间复杂度分析。
2. 代码在绝大多数情况下是低级别复杂度，只有极少数情况是高级别复杂度，并且具有一定的时序规律，则用均摊时间复杂度分析。