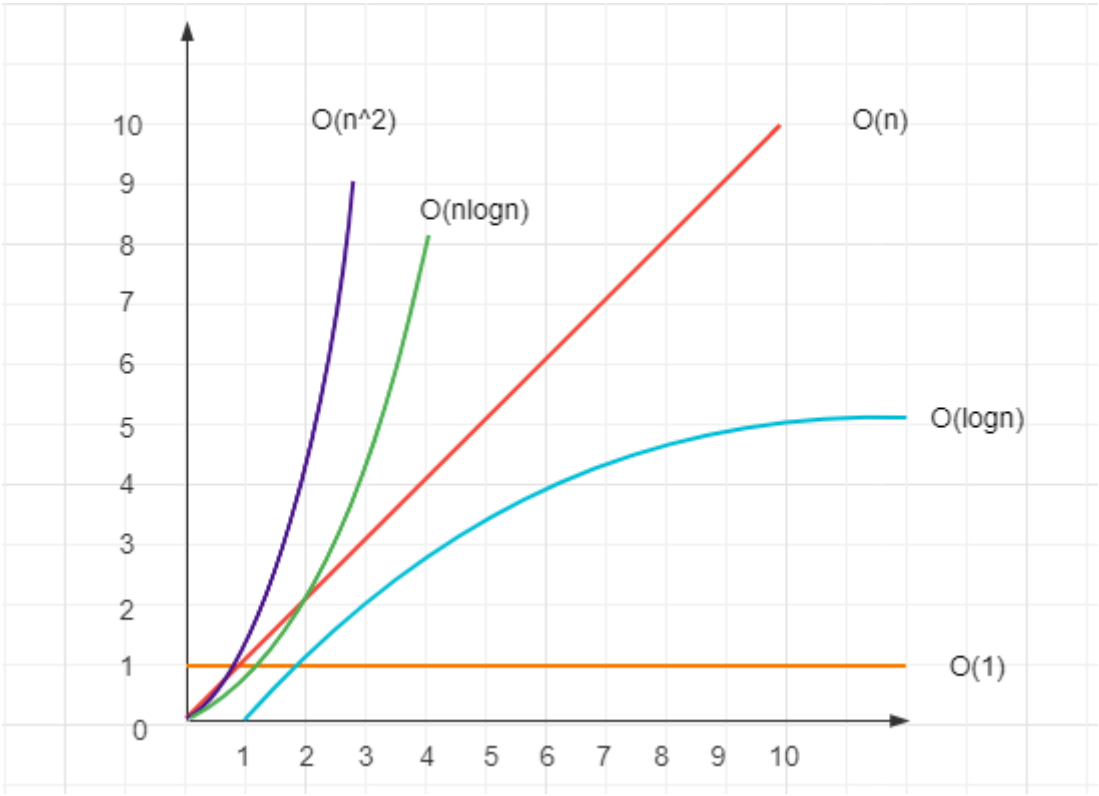
算法时间复杂度



O(1)

一行一行的书写代码，平铺直叙，没有循环。

O(logn)

有循环，但是使用了二分法，比如：二分查找算法。

二分法极大的减少了时间复杂度，计算量越大，减少越明显。

O(n)

有一层循环。

O(nlogn)

两层循环，有一层普通的循环，和一层二分算法的循环。例如：快速排序算法。

快速排序算法通过多次比较和交换来实现排序，其排序流程如下：

1. 首先设定一个分界值，通过该分界值将数组分成左右两部分。
2. 将大于或等于分界值的数据集中到数组右边，小于分界值的数据集中到数组的左边。此时，左边部分中各元素都小于分界值，而右边部分中各元素都大于或等于分界值。
3. 然后，左边和右边的数据可以独立排序。对于左侧的数组数据，又可以取一个分界值，将该部分数据分成左右两部分，同样在左边放置较小值，右边放置较大值。右侧的数组数据也可以做类似处理。

4. 重复上述过程，可以看出，这是一个递归定义。通过递归将左侧部分排好序后，再递归排好右侧部分的顺序。当左、右两个部分各数据排序完成后，整个数组的排序也就完成了。

```
const quickSort = (array) => {
    const sort = (arr, left = 0, right = arr.length - 1) => {
        if (left >= right) {//如果左边的索引大于等于右边的索引说明整理完毕
            return;
        }
        let i = left;
        let j = right;
        const baseVal = arr[j]; // 取无序数组最后一个数为基准值
        while (i < j) {//把所有比基准值小的数放在左边，大的数放在右边
            while (i < j && arr[i] <= baseVal) { //找到一个比基准值大的数交换
                i++;
            }
            arr[j] = arr[i]; // 将较大的值放在右边如果没有比基准值大的数就是将自己赋值给自己（i 等于 j）
            while (j > i && arr[j] >= baseVal) { //找到一个比基准值小的数交换
                j--;
            }
            arr[i] = arr[j]; // 将较小的值放在左边如果没有找到比基准值小的数就是将自己赋值给自己（i 等于 j）
        }
        arr[j] = baseVal; // 将基准值放至中央位置完成一次循环（这时候 j 等于 i ）
        sort(arr, left, j-1); // 将左边的无序数组重复上面的操作
        sort(arr, j+1, right); // 将右边的无序数组重复上面的操作
    }
    const newArr = array.concat(); // 为了保证这个函数是纯函数拷贝一次数组
    sort(newArr);
    return newArr;
}
```

O(n*2)

两层普通循环。比如：冒泡排序算法。

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
3. 针对所有的元素重复以上的步骤，除了最后一个。
4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

```
function bubbleSort(arr) {
    let i = arr.length, j;
    while (i > 0) {
        for (j = 0; j < i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                [arr[j + 1], arr[j]] = [arr[j], arr[j + 1]]; // 交换这两个数位置
            }
```

```
            }
            i--;
        }
        return arr;
    }

    let arr = [3, 2, 4, 9, 1, 5, 7, 6, 8];
    let arrSorted = bubbleSort(arr);
    console.log(arrSorted);
```