WEEK1 内容：

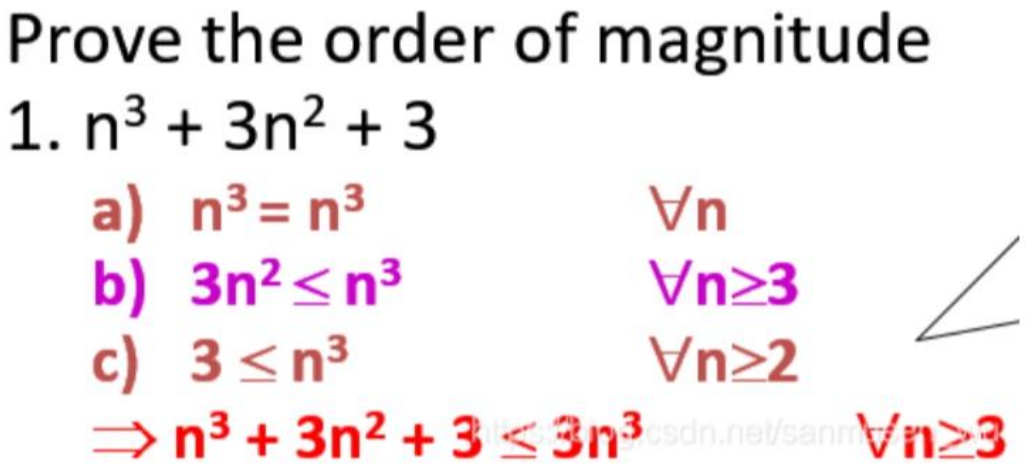
时间复杂度（Time complexity）
要注意的是 f(n)和大 O(g(n)), f(n)=O(g(n))

算法频度 f(n)：该算法基本操作需要执行的次数
辅助函数 g(n): n 取无穷时可近似 f(n) （n 趋近无穷时，lim f(n)/g(n) 等于一个常数）
时间渐进复杂度 O(g(n)): 时间复杂度，通常用 O(n)表示运行算法的规模，在代码分析中主要取决于循环语句执行次数。
下图展示了时间复杂度之间的大小比较关系

$$O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(2^n)$$

部分题型：

给定一式子，要求求解时间复杂度；证明各项在某个 n 值时恒小于某一项（与第三周内容类似但是存在差异），同样用来自 PPT 的图片进行阐述



week2 评估基础查找与排序算法

线性查找（Linear Search）
从头到尾查，O(n)

二分法查找（Binary Search)
原数组已排序时可用该算法，确定数组的左界和右界，每次查找查两界中间的数，如果想找的数比中间的数大，把中间＋1 更新为左界；反之，如果想找的数比中间的数小，把中间-1 更新为右界，O(log n)

寻找连续子串出现位置 （Search for a pattern)
从子串第一个字符位置依次向后比较,出现不相同的字符就把整体子串往后移一

位继续从子串第一个字符开始向后比，O(nm)

选择排序 （Selection Sort）
每次从原始数组中选择最小数，append 到新数组中，O(n^2)

冒泡排序（Bubble Sort）
从前到后，将原数组的相邻两数进行比较换位，直到数组末尾，此时末尾的数字已确定，下次（第 2 次）循环只需比较 0 到 n-1 个数字，以此类推，在第 i 次循环只需比较至 n-i＋1 个数，直到 i 等于 n 结束循环，O(n^2)

插入排序 （Insertion Sort Algorithm）选读
顺序遍历原始数组中的数，插入新数组中，将其放置于新数组里小于该数的数字与大于该数的数字之间，O(n^2)
在这里我认为使用频率较大的是冒泡排序和二分查找，二分在第三周有涉及故这里不多说，下面我用一个例题一下冒泡排序的代码与思想
一个句子指的是一个序列的单词用单个空格连接起来，且开头和结尾没有任何空格。每个单词都只包含小写或大写英文字母。我们可以给一个句子添加从 1 开始的单词位置索引，并且将句子中所有单词打乱顺序。
比方说，句子"This is a sentence"可以被打乱顺序得到"sentence4 a3 is2 This1"或者"is2 sentence4 This1 a3"。给你一个打乱顺序的句子 s,它包含的单词不超过 9 个，请你重新构造并得到原本顺序的句子。

```
class Solution {
    public String sortSentence(String s) {
        String[] arr = s.split(" ");
        int length = arr.length;
        String[] sentence = new String[length];
        for (int i = 0; i < length; i++) {
            int wordLength = arr[i].length();
            String word = arr[i].substring(0, wordLength - 1);
            int index = arr[i].charAt(wordLength - 1) - '0' - 1;
            sentence[index] = word;
        }
        StringBuffer sb = new StringBuffer();
        for (String word : sentence) {
            if (sb.length() > 0) {
                sb.append(" ");
            }
            sb.append(word);
        }
        return sb.toString();
    }
}
```

在以上题目中我们可以看到，我们首先外层循环，遍历所有单词，然后进行内层循环进行冒泡排序，即每次循环比较相邻的元素。如果第一个元素比第二个元素大，就交换它们的位置。 这样我们就可以确定一个数的正确位置，然后对每个数都重复该操作，最终得到我们想要的顺序。但是这种算法模式在数据较大或者说数据处理次数较多的时候不建议使用，因为它实际上时间复杂度很大。