WEEK3 INT102 NOTE

核心目标：学习分治法

一、Learning outcome：了解分治思想的工作原理，可以通过求解递归分析分治法的复杂性。并且了解部分分治示例的运行原理。
二、分治算法的思想基础：问题的分解，将问题分成几个相同问的的较小实例，运用递归进行解决并合并其解决方案，由此得到原本问题的解。
三、部分示例
1、二分查找
二分查找的基本步骤如下
选择一个基准元素，通常为数组的中间元素。

如果基准元素等于目标值，则查找成功，返回该元素下标。

如果目标值小于基准元素，则在数组的左半部分继续查找。

如果目标值大于基准元素，则在数组的右半部分继续查找。

重复步骤 2-4，直到找到目标元素或搜索范围为空

以下为一些个人代码示范，使用 JAVA 作为表示代码

```
private static int recursiveBinarySearch(int[] nums, int low, int high, int target) {

    if (low > high) {

        return -1;

    }

    int mid = low + ((high - low) >> 1);

    if (nums[mid] == target) {

        return mid;

    } else if (nums[mid] > target) {

        return recursiveBinarySearch(nums, low, mid - 1, target);

    } else {

        return recursiveBinarySearch(nums, mid + 1, high, target);

    }
```

}

同理它可以用递归来实现，但这里不再赘述

四、递归算法：

计算递归算法的时间复杂度

此时我们需要假设他为多少（通常考试我们只需要证明，他会给出对应的猜想）

证明流程如下：以例题为例

$$T(n) = \begin{cases} 1 & \text{if } n=1 \\ T(n/2) + 1 & \text{otherwise} \end{cases}$$

此时题设为 T(n)<=2logn，

则我们可以假设对于所有的 n’<n，都有 T（n/2）+1<=2log(n/2)+1

此时我们可先陈列特殊情况，这里为　　　　　n=1；

然后，我们对常规情况进行讨论，则有 T(n)=T(n/2)+1<=2log(n/2)+1=2(logn-1)+1<2logn

由此我们求解完成并得到他的时间复杂度。

课堂上另外两个练习我取其一进行讲解（流程相同）

Example

Prove that $\boxed{T(n) = \begin{cases} 1 & \text{if } n=1 \\ 2T(n/2) + 1 & \text{otherwise} \end{cases}}$ is O(n)

Guess: $T(n) \le 2n - 1$

对于这个题，我们首先参照例题对他的题设进行假设，得到对于所有的 n 都存在 t（n/2）<=n-1;

然后，我们将他带入到题设中，得到对于所有的 T（n）都有 T（n）<=2n-2+1=2n-1;

由此我们可以得到他的上边界就是 2n-1，根据我们之前学习的大 o 判定的规则就可以求解出在这一 guess 的提设要求下他的时间复杂度为 O（n）。

以上，我对我们求取递归复杂度的情况做一个小结：首先找到题设的要求，然后带入，按照高中的不等式知识进行上边界的求解，根据这一上边界来进行具体的时间复杂度的计算。

根据递归，我们可以猜测数量级如下

$$T(n) = T(n/2)+1 \quad T(n) \text{ is } O(\log n)$$

$$T(n) = 2T(n/2)+1 \quad T(n) \text{ is } O(n)$$

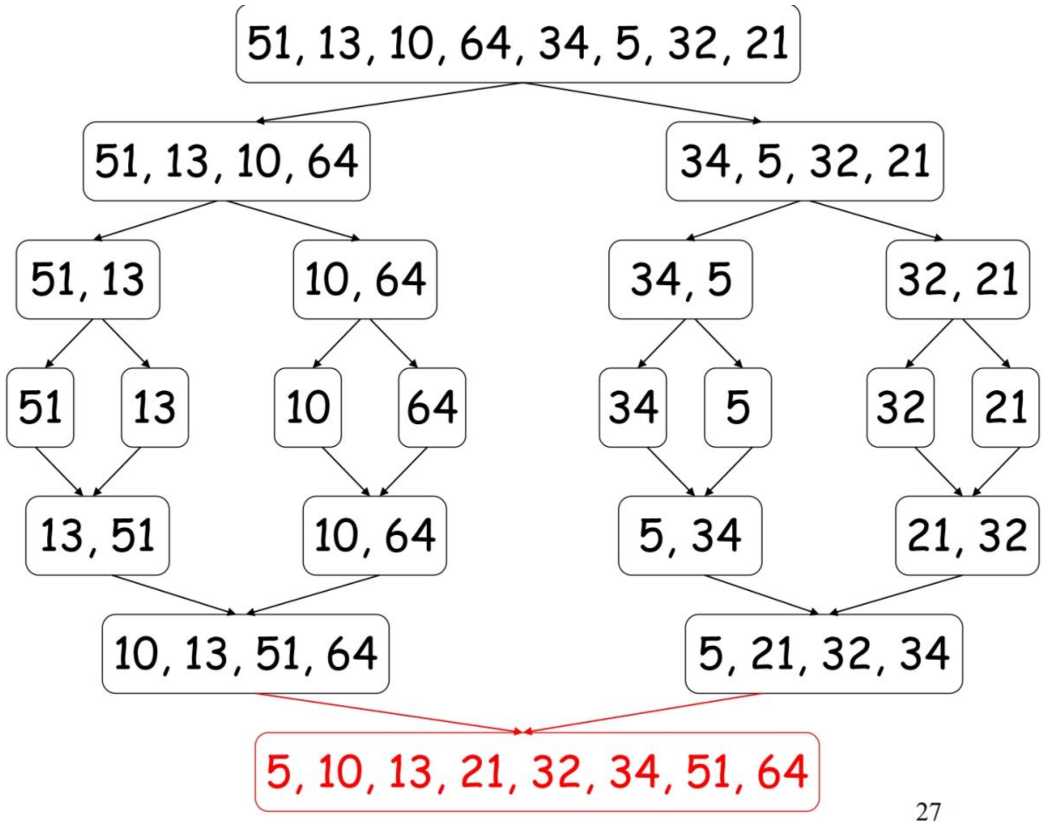
$$T(n) = 2T(n/2)+n \quad T(n) \text{ is } O(n \log n)$$

五、归并排序

归并排序是分治思想的体现，它主要通过如下方式实现

1、分解，将需要排序的数列不断分解至只存在一个数
2、将所有的子序列先成对比较排序再合并成对
3、此时重复步骤 2，最终得到对应的序列。

以图示内容为例



其中最后一步合并的具体步骤我认为可以表示如下：首先，我们将其中最小的两个数进行比较，将小者放置在前，然后把第一次比较剩余的数和另外一组第二小的比较并放置，以此类推进行遍历，最后得到排序的内容。

对于图示流程我们可以使用代码进行理解

```
def mergeSort(arr):
    if len(arr) > 1:  # 如果数组中的元素多于一个


        mid = len(arr)//2  # 找到数组的中间位置
        L = arr[:mid]  # 将数组元素分为两半
        R = arr[mid:]


        mergeSort(L)  # 对第一半进行递归调用
        mergeSort(R)  # 对第二半进行递归调用


        i = j = k = 0


        # 将数据复制到临时列表 L[]和 R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
```

```
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    # 检查是否有剩余的元素
    while i < len(L):
        arr[k] = L[i]
        i += 1
        k += 1

    while j < len(R):
        arr[k] = R[j]
        j += 1
        k += 1
```

以上代码为 Python 代码，但是相对来说我认为还是比较直观的。

这是代码在运行时的 tracetable（来自教案）

B: 10, 13, 51, 64　　　　C: 5, 21, 32, 34

	i	j	k	A[ ]
Before loop	0	0	0	empty
End of 1st iteration	0	1	1	5
End of 2nd iteration	1	1	2	5, 10
End of 3rd	2	1	3	5, 10, 13
End of 4th	2	2	4	5, 10, 13, 21
End of 5th	2	3	5	5, 10, 13, 21, 32
End of 6th	2	4	6	5, 10, 13, 21, 32, 34
				5, 10, 13, 21, 32, 34, 51, 64

最后，让我们再来看一下归并算法的时间复杂度

首先我先说明，他的时间复杂度是恒定为O（nlogn）的，原因如下：

我们可以先假设数列长度为N，那么我们将其分解为小数列则需要logN次，在这之后我们又会进行一个遍历并合并数列的操作，此时则会附加上一个O（N），将二者直接相乘我们就可以得到他的时间复杂度，无论情况好坏，这也是为什么说归并排序优于冒泡排序和快速排序。