8.O(nlogn)排序

1.归并排序

特性：
1.非原地空间复杂度O(n)、稳定排序算法
2.最好、最坏、平均时间复杂度都是O(nlogn)

归并的过程理解，其本质是一个递归过程
递推公式：merge_sort(p...r) = merge(merge_sort(p...q), merge_sort(q+1...r))
终止条件：p >= r 不用再继续分解

T(n)归并花费的总时间　T(n/2)表示子问题花费的时间 n表示合并的时间

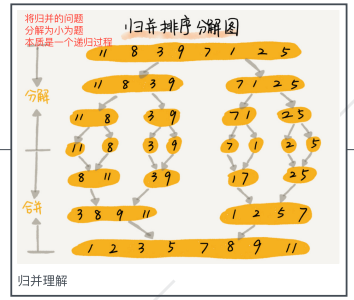
```
T(n) = 2*T(n/2) + n
     = 2*(2*T(n/4) + n/2) + n = 4*T(n/4) + 2*n
     = 4*(2*T(n/8) + n/4) + 2*n = 8*T(n/8) + 3*n
     = 8*(2*T(n/16) + n/8) + 3*n = 16*T(n/16) + 4*n
     ......
     = 2^k * T(n/2^k) + k * n
```

k为递归次数　当 T(n/2^k)=T(1) 时，n/2^k=1，我们得到 k=log2n　带入

```
= 2^log2(n) * T(1) + log2(n)*n
=n*T(1)+nlog2(n) //T(1)为常数执行时间
=Cn+nlog2(n)
```

时间复杂度求解过程

空间复杂度求解过程
递归代码的空间复杂度不能简单累加。每次合并操作都需要申请额外的内存空间，在合并完成之后，临时开辟的内存空间就被释放掉了。
在任意时刻，CPU 只会有一个函数在执行，也就只会有一个临时的内存空间在使用。临时内存空间最大也不会超过 n 个数据的大小，所以空间复杂度是 O(n)



归并理解

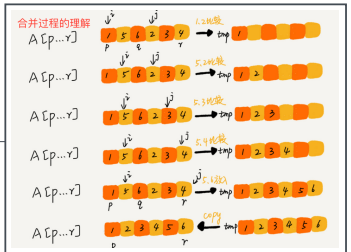
```
// 归并排序算法，A是数组，n表示数组大小
merge_sort(A, n) {
  merge_sort_c(A, 0, n-1)
}

// 递归调用函数
merge_sort_c(A, p, r) {
  // 递归终止条件
  if p >= r  then return

  // 取p到r之间的中间位置q
  q = (p+r) / 2
  // 分治递归
  merge_sort_c(A, p, q)
  merge_sort_c(A, q+1, r)
  // 将A[p...q]和A[q+1...r]合并为A[p...r]
  merge(A[p...r], A[p...q], A[q+1...r])
}
```

归并排序伪代码　递归过程　合并过程

归并伪代码

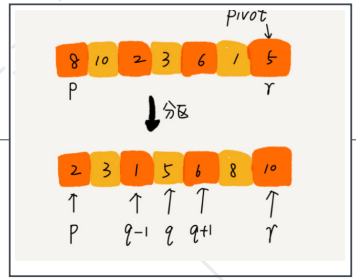


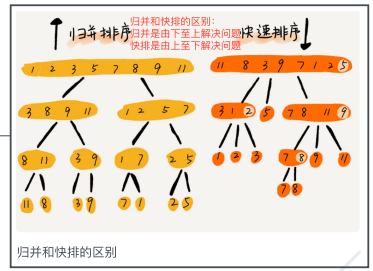
合并过程的理解

2.快速排序

特性：
1.原地、非稳定排序
2.时间复杂度O(nlogn)，极端条件下会退化到O(n^2),空间复杂度O(1)

快排的过程理解，本质也是递归+分而治之
递推公式：quick_sort(p...r) = quick_sort(p...q-1) + quick_sort(q+1... r)
终止条件：p >= r





归并和快排的区别

3.归并和快排练习

1. 找出第k大的数，要求O(n)
2. 10个文件，每一个单独有序，内存1g，将10个文件合并后有序