一.分治算法基础知识

1.引言

- 一个分治算法把问题实例划分成若干子实例(多数情况是分成两个)，并分别递归地解决每个子实例；然后把这些子实例的解组合起来，得到原问题实例的解；
- 实例：寻找最大最小值
  - 问题描述：在一个整数组A[1..n]中，同时寻找最大值和最小值。为了简化问题，不妨假定n是2的整数幂。
  - 基本思路：遍历数组，更新最大值和最小值，

    ```
    x=A[1];    y=A[1]
    for i=2 to n
        if A[i]<x then x=A[i]
        if A[i]>y then y=A[i]
    end for
    return (x,y)
    ```

    - 元素比较次数为2n-2
  - 分而治之：分别找到前半部分和后半部分的最大最小值，并进行比较

    ```
    struct result{
        int max;
        int min;
    };
    result MaxMin(vector<int> &nums,int begin,int end){
        result final_result;
        if(begin==end){
            final_result.max = nums[begin];
            final_result.min = nums[begin];
        }else{
            int mid = (begin+end)/2;
            result left_result = MaxMin(nums,begin,mid);
            result right_result = MaxMin(nums,mid+1,end);
            final_result.max = max(left_result.max,right_result.max);
            final_result.min = min(left_result.min,right_result.min);
        }
        return final_result;
    }
    ```

    - 时间复杂度分析：
      - 分治算法复杂度往往通过递归方程求解
      - T(n)代表元素比较次数
      - $$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\frac{n}{2}) + 2 & n > 1 \end{cases}$$

- 解得：T(n)=2n-2
- 如果将递归出口改为`begin=end-1`,将减少递归次数，T(n)为3n/2-2，可以有效减少比较次数

2.归并排序

- 基本思想：对前半段数组进行归并排序，对后半段数组进行归并排序，然后利用元素大小依次比较，将两段数组按正确的顺序合并为一个数组

```cpp
void MergeSort(vector<int> &nums,int begin,int end){
    if(begin>=end) return;
    int mid = (begin+end)/2;
    MergeSort(nums, begin, mid);
    MergeSort(nums, mid+1, end);
    int i=begin;
    int j=mid+1;
    int k=begin;
    while(i<=mid&&j<=end){
        if(nums[i]<nums[j]){
            extraSpace[k]=nums[i];
            i++;
        }else{
            extraSpace[k]=nums[j];
            j++;
        }
        k++;
    }
    if(i>mid){
        while(j<=end){
            extraSpace[k] = nums[j];
            j++;k++;
        }
    }else if(j>end){
        while(i<=mid){
            extraSpace[k] = nums[i];
            i++;k++;
        }
    }
    for(k=begin;k<=end;k++){
        nums[k] = extraSpace[k];
    }
}
```

- 算法分析：
  - $$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\frac{n}{2}) + \frac{n}{2} & n > 1 \end{cases}$$

$$\text{最小比较次数} \quad T(n) = \frac{nlogn}{2}$$

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\frac{n}{2}) + n - 1 & n > 1 \end{cases}$$

$$\text{最大比较次数} \quad T(n) = nlogn - n + 1$$

- 一道关于归并排序的题：数组中的逆序对
    - https://leetcode-cn.com/problems/shu-zu-zhong-de-ni-xu-dui-lcof/
    - 基本思路：逆序对由三种情况构成：
        - 数组前半段的逆序对
        - 数组后半段的逆序对
        - 数组前半段某个数字大于数组后半段某个数字

    关键是求解第三种情况，最直接的想法是遍历前半段数组，每个数字和后半段数组所有数字依次比较，统计出逆序对，但这样后半段数组就被多次遍历，不能有效利用已知的相对大小关系，可以对前后半段分别排序，这样就能免去与已知比自己小的数据的比较，递归+分别排序，自然想到了归并排序

```
class Solution {
public:
    vector<int> tmp;
    int reversePairs(vector<int>& nums) {
        tmp = nums;
        return MergeSort(nums,0,nums.size()-1);
    }
    int MergeSort(vector<int> &nums,int begin,int end){
        if(begin>=end) return 0;
        int mid = (begin+end)/2;
        int lf = MergeSort(nums,begin,mid);
        int rt = MergeSort(nums,mid+1,end);
        int k=begin,i=begin,j=mid+1;
        int sum=0;
        while(i<=mid&&j<=end){
            if(nums[i]<=nums[j]){
                tmp[k] = nums[i];
                sum += j-mid-1;
                k++;i++;
            }else{
                tmp[k] = nums[j];
                k++;j++;
            }
        }
        if(i>mid){
            while(j<=end){
                tmp[k] = nums[j];
                k++;j++;
            }
        }else if(j>end){
            while(i<=mid){
                tmp[k] = nums[i];
                sum += end-mid;
```

```
                k++;i++;
            }
        }
        for(k=begin;k<=end;k++){
            nums[k] = tmp[k];
        }
        return lf+rt+sum;
    }
};
```

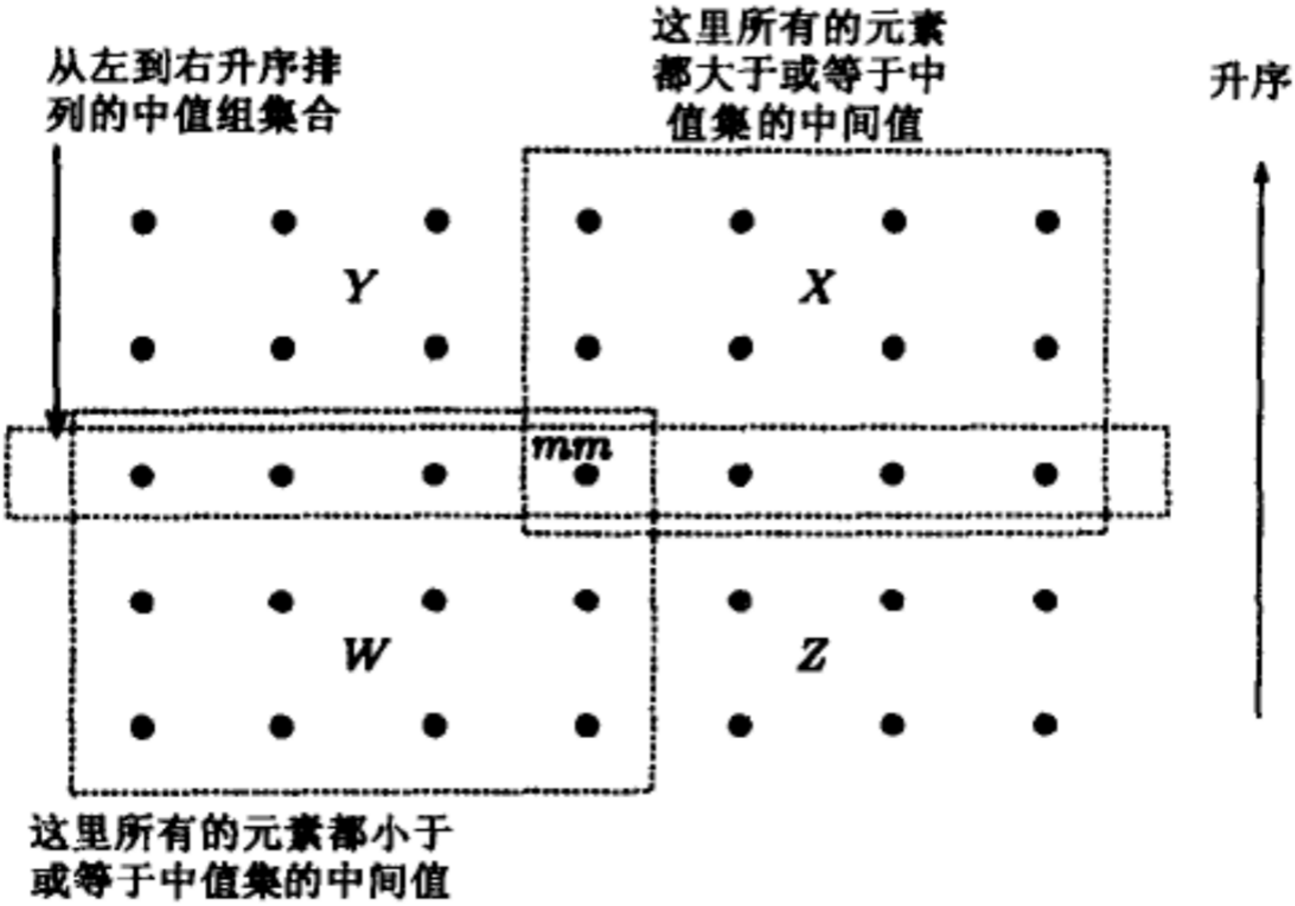
3.寻找第k小(大)元素

- 问题描述：在一个包含n个元素的集合中寻找第k个最小元素。
- 可以使用快排和堆（见`堆.pdf`），这里我们考虑分治算法
- SELECT算法：
    - 分治算法降低时间复杂度的本质：在二分搜索算法中，遍历数组的时间复杂度为`O(n)`,使用二分搜索可以根据中间值丢弃一半的数组，相当于减少了问题规模，所以最终的时间复杂度减少了（O(logn)），假设递归算法中，在每个递归调用的划分步骤后，我们丢弃元素的一个固定部分并且对剩余的元素递归。则问题的规模以几何级数递减，也就是在每个调用过程中，问题的规模以一个常因子被减小，最终形成一个几何级数，这个级数和小于n（假设某个算法丢弃1/3并对剩余的2/3部分递归）

$$cn + (\frac{2}{3})cn + (\frac{2}{3})^2cn + ... + (\frac{2}{3})^jcn + ... = \sum_{j=0}^{\infty} cn(\frac{2}{3})^j = 3cn = O(n)$$

    - 那如果我们在寻找第k小元素的过程中，每次都能排除一些一定比第k个元素大或小的元素，就可以减少搜索问题的规模
- 基本思想：
    - 首先，如果元素个数小于`预定义的阈值44`，则算法使用直接的方法计算第k小的元素，至于如何选择阈值在以后的算法分析中将会清楚。
    - 下一步把n个元素划分成n/5组，每组由5个元素组成，如果n不是5的倍数，则排除剩余的元素，这应当不影响算法的执行。每组进行排序并取出它的中项即第三个元素
    - 接着将这些中项序列中的中项元素记为mm，它是通过递归计算得到的。将数组A中的元素划分成三个数组: A1, A2, A3，其中分别包含小于、等于和大于mm的元素。
    - 最后，求出第k小的元素出现在三个数组中的哪一个。k<=|A1|，则在A1上递归，如果|A1|<k<=|A1+A2|，则返回mm，如果k>|A1+A2|，则修改k=k-|A1+A2|,在A3上递归。

- 原理说明：



从左到右升序排列的中值组集合
这里所有的元素都大于或等于中值集的中间值
升序
Y
X
mm
W
Z
这里所有的元素都小于或等于中值集的中间值

  - 阈值问题：

1. A_1' 表示所有小于等于 mm 的元素集合

   A_3' 表示所有大于等于 mm 的元素集合.

2. $|A_1'| \geq |W| \geq 3\left\lceil \lfloor \frac{n}{5} \rfloor /2 \right\rceil \geq \frac{3}{2}\lfloor \frac{n}{5} \rfloor$

   $|A_3| = n - |A_1'| \leq n - \frac{3}{2}\left(\frac{n-4}{5}\right) = 0.7n+1.2$

   $|A_1| \leq 0.7n + 1.2$

   说明一次递归后问题规模最多为 $0.7n+1.2$

   设我们预计问题规模减小到 cn

   解 $0.7n+1.2 \leq cn$.

   $$n \geq \frac{1.2}{c-0.7} \qquad 令 c=0.75,\ n \geq 44$$

   c 越小 ($c>0.7$). 阈值要求越大

- 算法实现：

```
SELECT
输入：n个元素的数组A[1..n]和整数k,1<=k<=n 。
输出：A中的第k小元素。
  1.select(A,1,n,k)

过程 select(A,low,high,k)
  1. p=high-low+1   // O(1)
  2. if p<44 then 将A排序 return (A[k])  // O(1)
  3. Let q= p/5.将A分成q组，每组5个元素。如果5不整除P，则排除剩余的元素   // O(n)
  4.将q组中的每一组单独排序，找出中项。所有中项的集合为M // O(n)
  5. mm=select(M,1,q q/2)           {mm 为中项集合的中项} // T(n/5)
  6. 将A[low..high] 分成三组: // O(n)
      A1={a|a<mm} A2={a|a=mm}  A3={a|a>mm}
  7. case   // T(0.75n)
     |A1| >=k: return select(A1,1,|A1|,k)
     |A1|+|A2| >=k: return mm
      |A1|+|A2|<k: return select(A3,1,|A3|,k-|A1|-|A2|)
  8. end case
```

$$T(n) \le \begin{cases} c & if\ n < 44 \\ T(\lfloor n/5 \rfloor) + T(\lfloor 3n/4 \rfloor) + cn & if\ n \ge 44 \end{cases}$$

$$T(n) \le \frac{cn}{1 - 1/5 - 3/4} = 20cn$$

综上：寻找第k小（大）元素的算法复杂度可以为O(n)

4.快速排序

- 基本思想：我们都知道升序的冒泡排序是每次将A[1...k]中元素最大值移动到A[k]，快速排序的基本思想与之类似，不过要求每次将某个元素的位置移动到排好序后该元素所在的位置，基本实现思想就是将小于该元素的放到该元素前，大于该元素的放到该元素后，这样就能找到该元素所在的位置，然后再对该元素前的数组和该元素后的数组进行递归，根据下述的算法实现，可以说明快速排序的平均时间复杂度为Θ(nlogn)
- 在最坏的情况下（数组已经为有序数组），算法QUICKSORT的运行时间是Θ(n^2)，然而如果总是选择中项作为主元，它的时间复杂性是Θ(nlogn)。

```
void quickSort(int a[],int begin,int end){
    if (begin>=end) return;
    int temp=a[begin];
    int i=begin;
    int j=end;
    while(i<j){
        while((a[j]>=temp)&&(i<j)) j--;
        a[i]=a[j];
        while((a[i]<temp)&&(i<j)) i++;
        a[j]=a[i];
    }
    a[i]=temp;
    quickSort(a,begin,i-1);
    quickSort(a,i+1,end);
}
```

5.矩阵乘法

- 问题描述：两个n*n矩阵相乘，需要做n^3次乘法和n^2(n-1)次加法，时间复杂度为Θ(n3).
- 递归方法：

假定n=2^k，$k \geq 0$，如果n≥2，则A，B和C可分别分成4个大小为n/2×n/2的子矩阵

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

用分治方法来计算C的组成:

$$C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{11} + A_{22}B_{22} \end{pmatrix}$$

$$T(n) = \begin{cases} 1 & n = 1 \\ 8T(\frac{n}{2}) + 4(\frac{n}{2})^2 & n >= 1 \end{cases}$$

算法复杂度数量级也是n^3，与上述算法相比只是矩阵元素相乘的次序不同

- STRASSEN算法
  - 基本思想：增加加法次数，减少乘法次数

我们首先计算以下的乘积

$$d_1=(a_{11}+a_{22})(b_{11}+b_{22})$$

$$d_2=(a_{21}+a_{22})b_{11}$$

$$d3=a_{11}(b_{12}-b_{22})$$

$$d_4=a_{22}(b_{21}-b_{11})$$

。

$$d_5=(a_{11}+a_{12})b_{22}$$

$$d_6=(a_{21}-a_{11})(b_{11}+b_{12})$$

$$d_7=(a_{12}-a_{22})(b_{21}+b_{22})$$

接着从一下面式子计算出C

$$C=\begin{pmatrix} d_1+d_4-d_5+d_7 & d_3+d_5 \\ d_2+d_4 & d_1+d_3-d_2+d_6 \end{pmatrix}$$

算法用到的加法是18次，乘法是7次，对于其运行时间产生以下递推式

$$T(n)=\begin{cases} m & if\ n=1 \\ 7T(n/2)+18(n/2)^2a & if\ n\geq 2 \end{cases}$$

或 $$T(n)=\begin{cases} m & if\ n=1 \\ 7T(n/2)+(9a/2)n^2 & if\ n\geq 2 \end{cases}$$

$$T(n)=(m+\frac{(9a/2)2^2}{7-2^2})n^{\log 7}-(\frac{(9a/2)2^2}{7-2^2})n^2=mn^{\log 7}+6an^{\log 7}-6an^2$$

即运行时间为:

$$\because(n^{\log 7})=\mathrm{O}(n^{2.81})$$

- 当元素个数不为2的幂时，找到大于等于矩阵row的最近的满足2的k次方的整数，用这个整数作为strassen矩

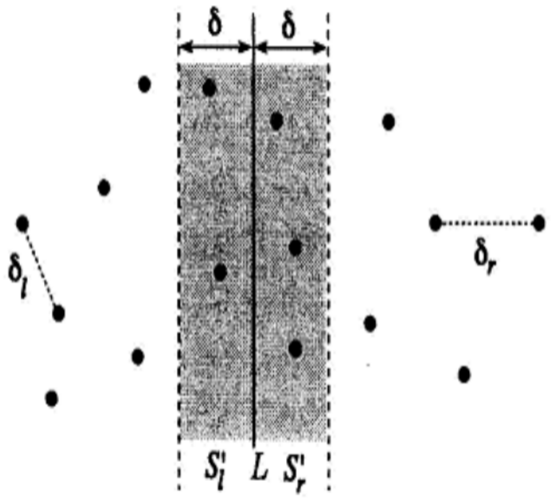
阵的strassenRow，填进去要计算的矩阵之后其余地方填0，在输出的时候只输出row×row的矩阵即可

6.最近点对问题

- 问题描述：设S是平面上n个点的集合，我们考虑在S中找到一个点对p和q的问题，使其相互距离最短。
- 分治方法：

  - Sort:
    - 第一步是以x坐标增序对S中的点排序
  - Divide:
    - S点集被垂直线L大约划分成两个子集S_l和S_r，使$|S_l| = \lfloor |S|/2 \rfloor$， $|S_r| = \lceil |S|/2 \rceil$，设L是经过x坐标$S[\lfloor n/2 \rfloor]$的垂直线，这样在$S_l$中的所有点都落在或靠近L的左边，而所有$S_r$中的点落在或靠近L的右边。
  - Conquer:
    - 现在按上述递归地进行，两个子集S_l和S_r的最小间距δ_l和δ_r可分别计算出来。
  - Combine:
    - 组合步骤是把在S_l中的点与S_r中的点之间的最小间距δ’计算出来。最后所要求的解是δ_l，δ_r和δ’中的最小值。

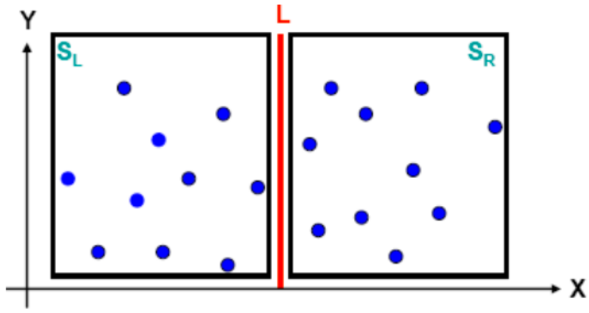
- 怎样合并结果，这一步的关键在于计算δ’，计算S_l中的每个点与S_r中每个点之间的距离的朴素方法需要$\Omega(n^2)$。



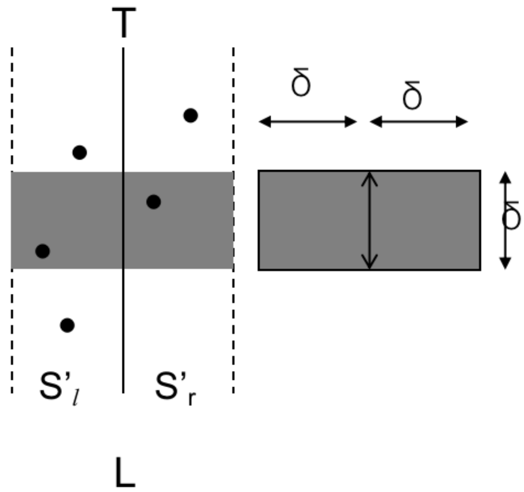
■设$\delta = \min\{\delta_l, \delta_r\}$，如果最近点对由$S_l$中的某个点$p_l$与$S_r$中的某个点$p_r$组成，则$p_l$和$p_r$一定在划分线L的距离δ内。

■因此，如果令S'_l和S'_r分别表示为在线L距离δ内的S_l和S_r中的点，则p_l一定在S'_l中，p_r一定在S'_r中。

- 假设$\delta' \le \delta$，则存在两点$p_l \in S'_l$和$p_r \in S'_r$，有$d(p_l, p_r) = \delta'$，从而p_l和p_r之间的垂直距离不超过δ



- 因为p_l, p_r这两点都在以垂直线L为中心的$\delta \times 2\delta$矩形区内或其边界上
- 设T是两个垂直带内的点的集合
- 如果在$\delta \times 2\delta$矩形区内，任意两点间的距离一定不低于δ，则这个矩形最多能容纳8个点，其中至多4个点属于S_l，4个点属于S_r。



二.习题

1.计算右侧小于当前元素的个数

- https://leetcode-cn.com/problems/count-of-smaller-numbers-after-self/
- 基本思路：这道题是在归并排序一节提到的例题的变体，对于任意元素i，都可以经过不断二分，计算出其后半段有多少元素小于i，每次二分都更新前半段的count[i]，经过不断二分后就可得出最终结果。由于在移动过程中元素的角标会发生变化，考虑增加一个数组，保存元素原始角标，随着元素的移动进行移动。

```
class Solution {
public:
    vector<int> tmp;
    vector<int> index;
    vector<int> tmp_index;
    vector<int> countSmaller(vector<int>& nums) {
        vector<int> counts(nums.size(),0);
        tmp = nums;
        for(int i=0;i<nums.size();i++){
            index.push_back(i);
            tmp_index.push_back(i);
        }
        MergeSort(nums,0,nums.size()-1,counts);
        return counts;
    }
```

```
    void MergeSort(vector<int> &nums,int begin,int end,vector<int> &counts){
        if(begin>=end) return;
        int mid=(begin+end)/2;
        MergeSort(nums,begin,mid,counts);
        MergeSort(nums,mid+1,end,counts);
        int k=begin,i=begin,j=mid+1;
        while(i<=mid&&j<=end){
            if(nums[i]<=nums[j]){
                tmp[k]=nums[i];
                tmp_index[k]=index[i];
                counts[index[i]]+=j-mid-1;
                i++;k++;
            }else{
                tmp[k]=nums[j];
                tmp_index[k]=index[j];
                k++;j++;
            }
        }
        if(i>mid){
            while(j<=end){
                tmp[k]=nums[j];
                tmp_index[k]=index[j];
                k++;j++;
            }
        }else{
            while(i<=mid){
                tmp[k]=nums[i];
                tmp_index[k]=index[i];
                counts[index[i]] += end-mid;
                k++;i++;
            }
        }
        for(k=begin;k<=end;k++){
            nums[k]=tmp[k];
            index[k]=tmp_index[k];
        }
    }
};
```

2.翻转对

- https://leetcode-cn.com/problems/reverse-pairs/
- 基本思路:
  - 题目是归并排序例题及前一题的变体
  - 首先，题目中说所有数字在32位整数表示范围内，如果使用先乘2再进行比较，可能溢出，因此需要对nums[i]类型转换为long
  - 先利用归并排序的思想，统计出`nums[i]>2*nums[j]`的个数，然后再进行正常排序

```
class Solution {
```

```
public:
    vector<int> tmp;
    int reversePairs(vector<int>& nums) {
        tmp=nums;
        return MergeSort(nums,0,nums.size()-1);
    }
    int MergeSort(vector<int>& nums,int begin,int end){
        if(begin>=end) return 0;
        int mid=(begin+end)/2;
        int counts=0;
        int lf = MergeSort(nums,begin,mid);
        int rt = MergeSort(nums,mid+1,end);
        int k=begin,i=begin,j=mid+1;
        while(i<=mid&&j<=end){
            if(nums[i]<=2*long(nums[j])){
                counts+=j-mid-1;
                i++;k++;
            }else{
                j++;k++;
            }
        }
        if(j>end){
            while(i<=mid){
                counts += end-mid;
                i++;k++;
            }
        }
        i=begin;j=mid+1;k=begin;
        while(i<=mid&&j<=end){
            if(nums[i]<=nums[j]){
                tmp[k]=nums[i];
                i++;k++;
            }else{
                tmp[k]=nums[j];
                j++;k++;
            }
        }
        if(j>end){
            while(i<=mid){
                tmp[k]=nums[i];
                i++;k++;
            }
        }else{
            while(j<=end){
                tmp[k]=nums[j];
                j++;k++;
            }
        }
        for(k=begin;k<=end;k++){
```

```
            nums[k]=tmp[k];
        }
        return lf+rt+counts;
    }
};
```

3.最大子序和问题

- https://leetcode-cn.com/problems/maximum-subarray/
- 基本思路：这是一道分治思想的经典题目，将最大子序和分为三种情况
  - 最大序列在数组前半段
  - 最大序列在数组后半段
  - 最大序列横跨数组前后半段（意味着最大序列必须取到a[n/2]和a[n/2+1]），这种情况最大序列值为`left+a[n/2]+a[n/2+1]+right`，只需分别从`n/2-1`和`n/2+2`开始向前或向后遍历，分别得到`left`和`right`最大值然后相加即可。

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        return maxSubArray2(nums,0,nums.size()-1);
    }
    int maxSubArray2(vector<int>& nums,int begin,int end){
        if(begin==end) return nums[begin];
        int mid=(begin+end)/2;
        int maxRes=nums[mid]+nums[mid+1];
        int temp=maxRes;
        for(int i=mid-1;i>=begin;i--){
            temp += nums[i];
            if(temp>=maxRes) maxRes=temp;
        }
        temp = maxRes;
        for(int i=mid+2;i<=end;i++){
            temp += nums[i];
            if(temp>=maxRes) maxRes=temp;
        }
        int lf=maxSubArray2(nums,begin,mid);
        int rt=maxSubArray2(nums,mid+1,end);
        if(lf>maxRes) maxRes=lf;
        if(rt>maxRes) maxRes=rt;
        return maxRes;
    }
};
```

树的问题