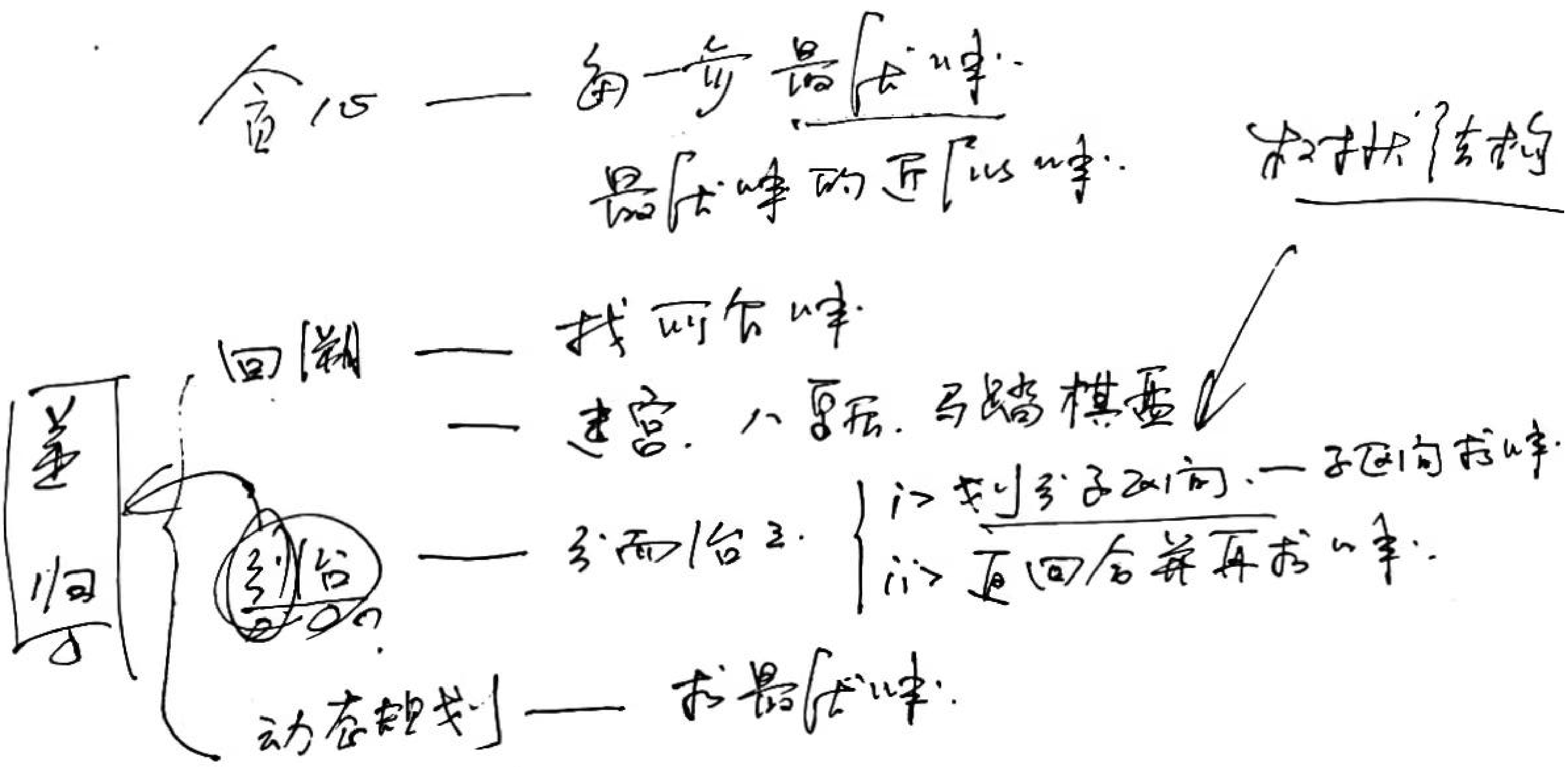
分治算法 —— 分而治之

步骤
- i> 划分子区间
  - ⇒ 解子区间求解
- ii> 合并求解


贪心 — 每一步最优解
最优解的近似解
树状结构
回溯 — 找所有解
— 迷宫、八皇后、马踏棋盘
递归
划分 合并
分而治之
1> 划分子区间 — 子区间求解
2> 再回合并再求解
动态规划 — 求最优解

归并排序：

[例]：

L　　　　　　　　M | M+1　　　　H

49　38　65　97 | 76　13　27

$M = \frac{L+H}{2}$

二分为两个子区间

[49　38　65　97]　[76　13　27]

[49　38]　[65　97]　[76　13]　[27]

[49]　[38]　[65]　[97]　[76]　[13]　[27]

① [38　49]　[65　97]　[13　76]　[27]

② [38　49　65　97]　[13　27　76]

③ [13　27　38　49　65　76　97]

```
fun(int a[], int L, int H)
{
    if(L<H)
    {
        M=(L+H)/2;
        fun(a, L, M);
        fun(a, M+1, H);
        合并
    }
}
```

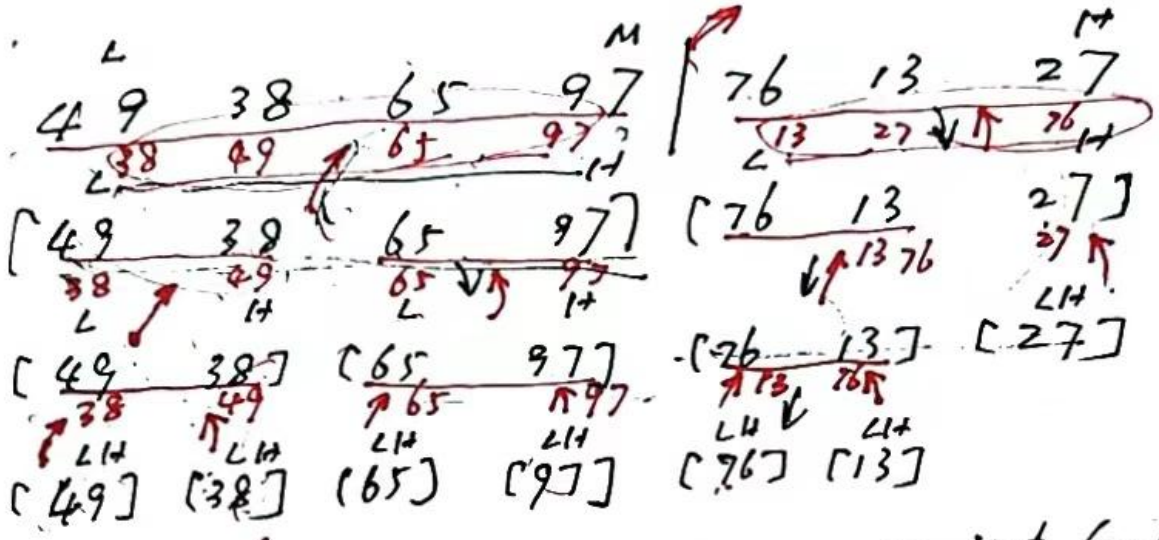
基础选择题：

[49] [38] [65] [97] [76] [13] [27]

① [38 49] [65 97] [13 76] [27]

② [38 49 65 97] [13 27 76]

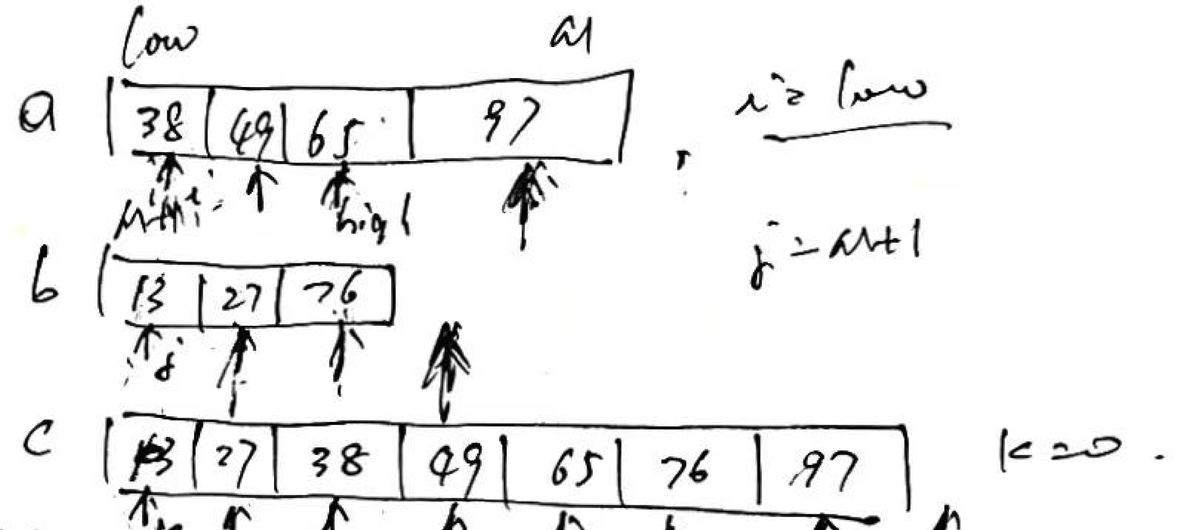
③ [13 27 38 49 65 76 97]



算法：

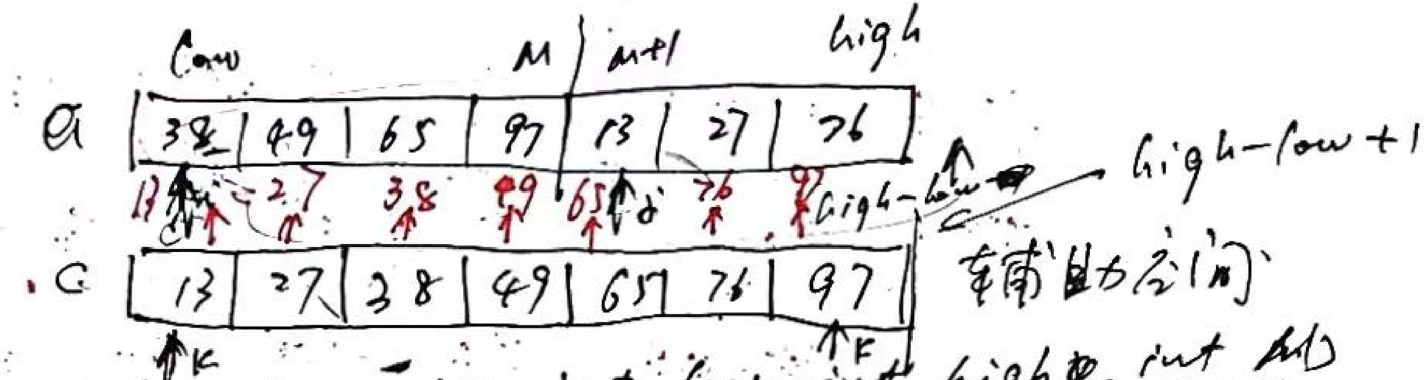
```
void MergeSort(int a[], int low, int high)
{
    int m;
    if(low<high){ m=(low+high)/2;
        ① 左=>: MergeSort(a, low, m);
        ② 右=>: MergeSort(a, m+1, high);
        ③ 合并排序 Merge(a, low, m, m+1, high);
    }
}
```

a[low..m] 和 a[m+1..high]




```
while(i<m+1 && j<high+1)
    if(a[low] < b[j])   Y: c[k++] = a[i++]
                        N: c[k++] = b[j++]
}
if(i<m+1){
    for(; i<m+1; ){
        c[k++]=a[i++];
    }
}
else{ for(; j<high+1; ){
        c[k++]=b[j++];
    }
}
```

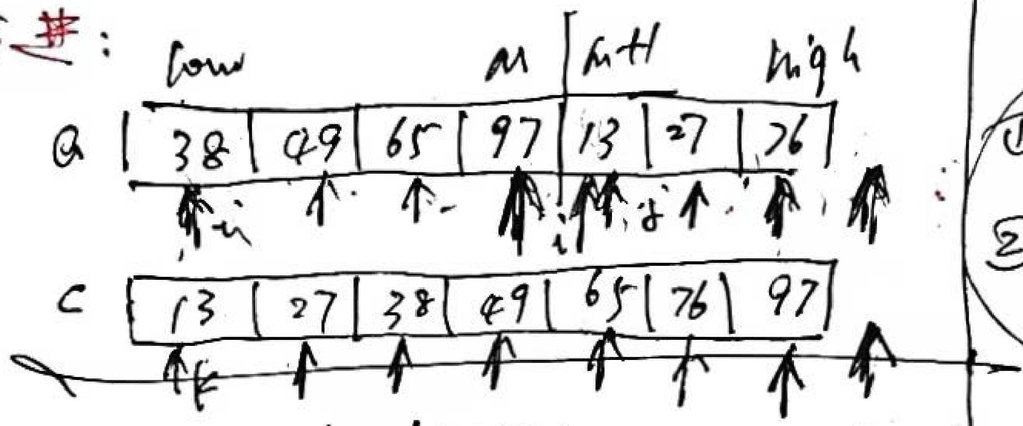
合并:




```
void Merge(int a[], int low, int high, int m)
{
    int *c;
    int i, j, k;
    c = (int *)malloc((high-low+1)*sizeof(int));
    i=low; j=m+1; k=0;
    while(i<m+1 && j<high+1){
        if(a[i]<a[j])
            c[k++]=a[i++];
        else
            c[k++]=a[j++];
    }
    if(i<m+1){ // j出去. i没完.
        for(; i<m+1; ){
            c[k++]=a[i++];
        }
    }
    else{ // i完了. j没完.
        for(; j<high+1; ){
            c[k++]=a[j++];
        }
    }
    // 将c中的值放回a.
    for(k=0, i=low; i<high+1; ){
        a[i++]=c[k++];
    }
    free(c);
}
```

两个子顺序表都没完.去比较

改进：




```
i=low; j=m+1; k=0;
while(i<m+1 || j<high+1){
    if(j>high || i<m+1 && a[i]<a[j]){
        c[k++]=a[i++];
    }
    else{
        c[k++]=a[j++];
    }
}
//将c[]复制回a中
```

讨论 a[i]

① b[j]访问完了

② b[j]没完 ⇒ a[i]也没访问完 && a[i]<b[j]

讨论 b[j]：除此以外的

```
if(j>high || i<m+1 && a[i]<b[j]){
    c[k++]=a[i++];
}
else{
    访问 a[j++];
}
```

搞清楚：

① 子表划分.

② 递归返回 ⟨ 下标：左 low — high ⇒ low ~ M.
右 low — high ⇒ M+1 — high
结果：两个子表已排好序

★ ③ 两个有序子表合并一个有序表

时间复杂度： $O(\log_2 n)$

空间复杂度： $O(n)$.

内排序：① 时间复杂度　　稳定性
空间复杂度
② 算法思想，排序方法，once的结果，
③ 代码描写。

① 平均时间性能：< 快排最佳
在最坏情况下不如堆排和归并。
☆在n值较大时 < 归并省时间，但辅助存储大

② 简单性：< 简单排序：直接插入，直接选择，冒泡
直接插入排序法往往和堆排，归并结合使用。

☆③ 基数排序：适合于n值较大，但关键字较小　　稳定✓

③ 稳定性 < 基数排序稳定，时间复杂度 $O(n^2)$ 往往比较
快排，希尔，堆排时间性能好，但不稳定。